DISTRIBUTED GRID ANALYTICS PLATFORM (DGAP) FOR POWER GRID MONITORING AT THE DISTRIBUTION LEVEL

BY

YUQI LI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Assistant Professor Robert C. N. Pilawa-Podgurski

ABSTRACT

Phasor measurement units (PMUs) which measure electrical waves with real-time synchronization at widely spread points across the power grid offer great benefits. While the PMU device concept is well known in the power industry, the field of power system analysis stands to benefit greatly from using different methods of designing and implementing an inexpensive PMU that can be widely and densely distributed on the grid. Traditional PMUs are mainly installed at the transmission level, where they are hard to install and maintain, and can be expensive due to the rating requirements of the components. Given their benefits and increasingly widespread installation, easier-to-maintain and less costly PMUs are desired.

In 2000, frequency disturbance recorders (FDRs), which are single-phase PMUs that monitor the power grid at the 120 V distribution level, were operated for the Frequency monitoring Network (FNET) project by Virginia Tech and the University of Tennessee. While installing FDRs at the low-voltage distribution level of the power grid was a great step toward reducing the cost and limitations of PMU use, there are still drawbacks and significant room for improvement: the sampling frequency is low at 1440 samples per second (SPS), there is no auxiliary power supply to support the device during an atypical power grid event, and the USD 2000 price can be driven lower.

This thesis introduces the Distributed Grid Analytics Platform (DGAP) which has a higher sampling rate (20k SPS), a backup power supply, smaller size, and much lower cost (USD 200) while keeping the functionality of the FDRs including accurate data acquisition, GPS time synchronization, internet connectivity, and open source data upload. The improvements were realized by a more succinct approach for the system design and more updated component selection, which will be explained in this thesis. The designed

DGAPs were built into prototypes and tested in household power outlets, experimentally validating their functionality.

To my family, for their endless love and support.

ACKNOWLEDGMENTS

Thanks to Professor Robert C.N. Pilawa-Podgurski, my thesis advisor, for his help and guidance during the past two years. He showed me how to be a professional and serious researcher and prepared me to enter the industrial world. I am grateful for the opportunity to learn from him not only as a great researcher but also as a great person in other aspects: I enjoyed his power electronics classes in which he always presented the contents in a practical but interesting way, and I received great advice from him throughout my graduate school and future planning, without which my career path could be very different.

Thanks to Andrew R. Stillwell for spending much of his precious time on my project. The work in this thesis would not be possible without his tremendous help. His industrial background and insight into micro-controllers were momentous to the success of the project. Many people helped me a lot with the work in this thesis. Christopher Drew helped looking into data transfer between C and Python as well as data upload to Open PDC. Mark Hirsbrunner coded the internet connection of Beaglebone. Rodrigo Serna and Jingyi Ma helped experimenting with ADC and GPS. Thanks to my fellow researchers in the research group, Christopher Barth, Enver Candan, Yutian Lei, Benjamin Macy, Shibin Qin, Joshua Sheridan, and many more, for sharing their knowledge and all their help.

Thanks to Professor Thomas J. Overbye for funding my project through the Illinois Center for a Smarter Electric Grid (ICSEG).

Thanks to those who have accompanied me throughout graduate school, especially to Yujia Zhang, the other girl in my power electronics lab during my first year, who has been my partner for so many lab courses as well as

for lunches; Di Fan, who has been like a sister to me, has great taste in life and great interest in burning money with me; and Jianan Wang, who almost did not make it into these acknowledgments since he showed up so late in my life, but thankfully he finally came.

Most of all, thanks to my parents for their support. Even though they are not near by my side most of the time, their help and love were a great comfort in my endeavors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Phasor measurement units (PMU) have been playing an important role in power grid monitoring. A PMU is a device that measures the voltage and current on an electricity grid and obtains the frequency and phase data from the voltage and current data measured [5]. The PMUs are also time synchronized, which enables synchronized real-time measurements of wide ranging measurement points on the grid [6]. With continuous sampled and synchronized real-time data from different locations, the data are widely used for power system model verification. More importantly, PMUs can be used to perform real-time power grid monitoring; if incorporated with relays and other circuit protective devices [7], the device can trip the grid to prevent damage before things happen [8]. From this real-time monitoring aspect, the data collected around an atypical event can be studied for power system behavior analysis and contribute to future power grid technologies [9]. Due to all the benefits a PMU can provide, they are considered one of the most important measuring devices in the future of power systems [10].

As technology has evolved, PMUs have evolved significantly as well [11]. Traditional PMUs are usually installed on transmission lines with voltage up to 765 kV. This PMU is very expensive because it must operate safely in a high voltage environment; in addition, installing a device on the transmission line can be expensive and time-consuming, costing tens of thousands of dollars per device and requiring several months of effort. Maintaining the device can also be costly. These constraints limit the number of PMUs we can distribute to the field and, thus, their use in the electric power industry.

In 2000, Frequency Disturbance Recorders (FDRs), which are PMUs that can be installed at the low-voltage distribution level of the power grid [12], were introduced. Since voltages involved are much lower than those for traditional PMUs, FDRs are relatively inexpensive and simple to install. FDRs sample at 1440 samples per second and transmit to the Frequency monitoring Network (FNET) for processing and storage. As the distribution-level FDRs have been developed, the number of deployed PMUs has significantly increased.

As time goes on, it seems that we always need PMUs to be distributed more densely and sample data faster. That is why Distributed Grid Analytics Platforms (DGAPs) have been developed. DGAPs aim to sample faster and operate more reliably while costing less. DGAPs have been developed and the prototypes are tested and ready for deployment in the field. The cost per DGAP is estimated to be under 200 USD, one tenth that of FDRs. The DGAP is implemented with backup battery power supply, so it can capture data even when the voltage at the power grid is too low to support the device. The DGAP samples data at 20k Hz, which capture more transient behavior of the power grid. The data collected by DGAP — the time-synchronized rms voltage, frequency, and phase data — can be used to do event detection and location estimation [13], power system event visualization, power grid oscillation detection [14], on-line trip detection, off grid/islanding detection [15], and all other power system research. As the data is more transient and more reliable, it could help advance power system research to a new future, especially when renewable energy sources such as solar power come into play [16] and more transient and accurate monitoring are needed.

The upper level description of how the DGAP works is as follows. The DGAP samples a scaled-down version of the power outlet's voltage signal using an analog-to-digital converter, and these data are processed to a microcontroller for further calculations. Each measurement is time stamped with time information provided by a GPS, and then all those data are transmitted to a server for processing and storage. More detailed explanations will be presented in this thesis.

1.2 Organization of this Thesis

This remainder of thesis is divided into three chapters.

Chapter 2 introduces the design and implementation of the DGAP hardware. Considerations during system design and component selections are described in detail. Advantages and possible future improvements of each component are highlighted. These prototypes are used for all the software design and experimental testing explained in the following chapters.

Chapter 3 explains the software structure of DGAP and mathematical implementations for obtaining RMS voltage, frequency, and phase data.

Chapter 4 presents the test setup and experiments performed to verify the functionality of the DGAP prototypes. Obtained data are analyzed, and future efforts of what need to be improved are discussed.

CHAPTER 2

HARDWARE IMPLEMENTATION

In this chapter, the design and implementation of the DGAP hardware are introduced. Considerations during system design and component selections are described in detail. Future improvements that can be done are stated in each section.

2.1 System Level Structure

The system level hardware structure of the DGAP is shown in Fig. 2.1

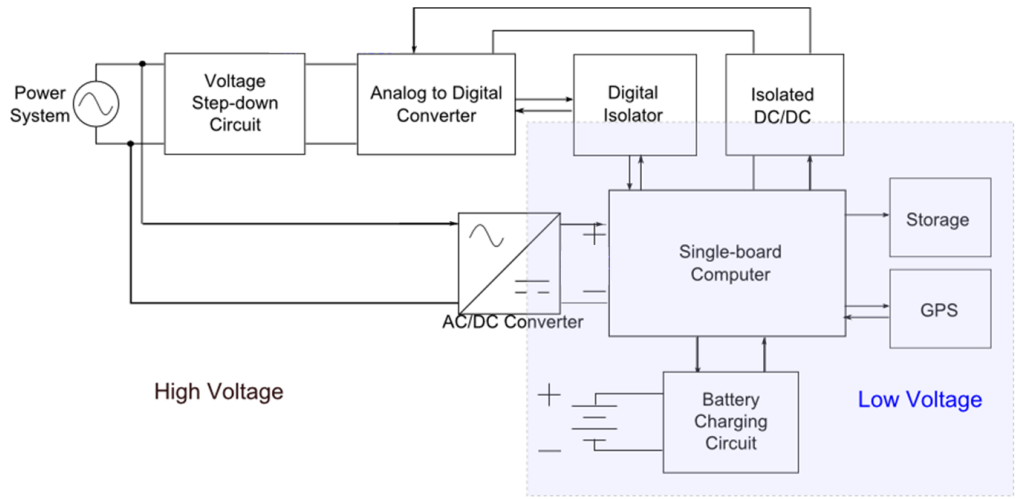

Figure 2.1: DGAP hardware structure at the system level.

The "Power System"terminal represents the 120 Vrms AC power outlet. From the power outlet, the wire goes in two directions: one goes to the data acquiring system, and the other provides power to the device.

At the beginning of the data acquiring system, the 120 Vrms AC voltage feeds into a voltage-step-down circuit through an AC power connector [17]. The voltage-step-down circuit converts the 120 Vrms AC voltage, which is in the range of ±170 V, to the range of 0 V to 3.3 V. This step is essential since most analog-to-digital converters (ADCs) only take positive analog voltage input under 5 V. The reason we step down the voltage even further to limit the maxim input value to 3.3 V is that the device is operating on a 5 V power rail, thus the reference voltage provided to the ADC cannot reach 5 V, which places an upper limit of the input voltage; this choice of values will be explained in detail in Section 2.2. The voltage-step-down circuit is shown in Fig. 2.2.

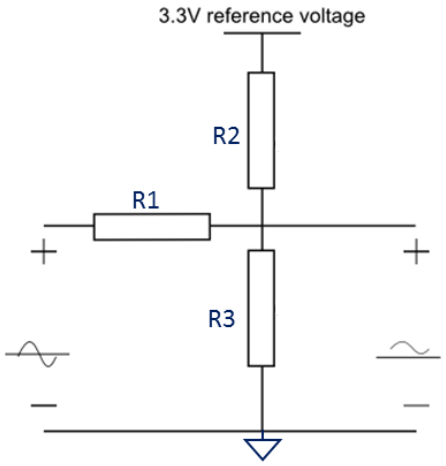

Figure 2.2: The voltage-step-down circuit.

At the input of the voltage-step-down circuit, an input range of ±185 V was used instead of the ±170 V, which is $\pm 120 \times \sqrt{2}$ V, to allow some room for disturbance and abnormal performance of the power grid; the output of the circuit ranged from 0.1 V to 3.2 V instead of 0 V to 3.3 V for the same purpose. With this setup, the relationship between input and output can be defined as

$$\begin{cases} output = 3.2\ V, & \text{when } input = 185\ V \\ output = 0.1\ V, & \text{when } input = -185\ V \end{cases} \tag{2.1}$$

By Kirchhoff's current law, which defines that at any node in an electrical

circuit, the sum of currents flowing into that node is equal to the sum of currents flowing out of that node, the circuit can be represented as

$$\begin{cases} \frac{3.3\ V-3.2\ V}{R_2} + \frac{185\ V-3.2\ V}{R_1} = \frac{3.2\ V}{R_3} \\ \frac{3.3\ V-0.1V}{R_2} + \frac{-185\ V-0.1V}{R_1} = \frac{0.1\ V}{R_3} \end{cases} \tag{2.2}$$

By solving Equation. 2.2, the ratio of R_1, R_2, and R_3 was obtained to be roughly 58:1:1. The choice of exact values of the resistors will be explained in Section 2.2.7.

The stepped-down voltage feeds into the ADC as analog signals, then the ADC outputs digital signals to the micro-controller through a digital isolator. As indicated in Fig. 2.1, there is a high voltage region, which has direct contact to the power outlet, and a low voltage region, which does not touch the 120 Vrms AC voltage at all. The function of the digital isolator is to isolate the high voltage and low voltage region so that if anything abnormal happens at the power grid, or if any component at the high voltage region does not function correctly, the micro-controller does not get burned due to possible overrated voltage or current. The isolated DC/DC in the power path between micro-controller power rail and the ADC serves the same purpose.

From another route, the power outlet feeds into a power adapter which provides 5V DC to the micro-controller, which provides power to the rest of the system. When the power outlet is incapable of providing power to the device, the micro-controller starts drawing power from an external battery through the battery charging circuit; once the device start drawing power from the power outlet, the battery charging circuit charges the rechargeable battery for future use.

A GPS is built in for real-time stamps and synchronization. The micro-controller is connected to internet via Ethernet for data upload. When the device temporarily loses internet connection, a single 256Mb x16 DDR3L 4GB (512MB) memory on the single-board computer will be used for data storage.

2.2 Component Selection

This section explains how the components in the system are selected. A list of components mentioned in this section can be found in Table D.1 within Appendix D.

2.2.1 Single-board Computer

When it comes to choosing what single-board computer to use, there are several different aspect to be considered and several functionalities to be fulfilled.

The single-board computer needs to have a micro-controller that can perform real-time data sampling at a high enough rate as well as efficiently analyze the acquired data mathematically. It needs to have output power rails to power the rest of the system. It has to have two or more communication interfaces to communicate with peripheral components such as GPS and ADC. It needs to have internet access through Ethernet or wi-fi or some other kind for data sharing. It needs to have some kind of data storage to store data temporarily. And also, since one of the goals of this DGAP is to make it as inexpensive as possible, it should be open-sourced with a reasonable price.

Beaglebone Black (BBB), as shown in Fig. 2.3, is one single-board computer which fulfills the listed requirements.

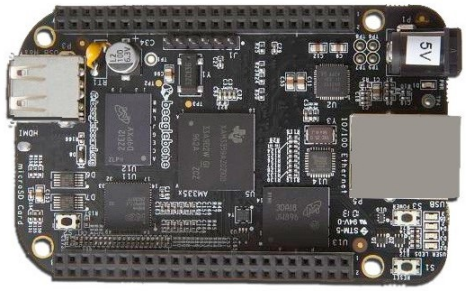

Figure 2.3: Top view of Beaglebone Black.

The functional block diagram of the Sitara XAM3359AZCZ100 Processor on board is shown in Fig. 2.4.

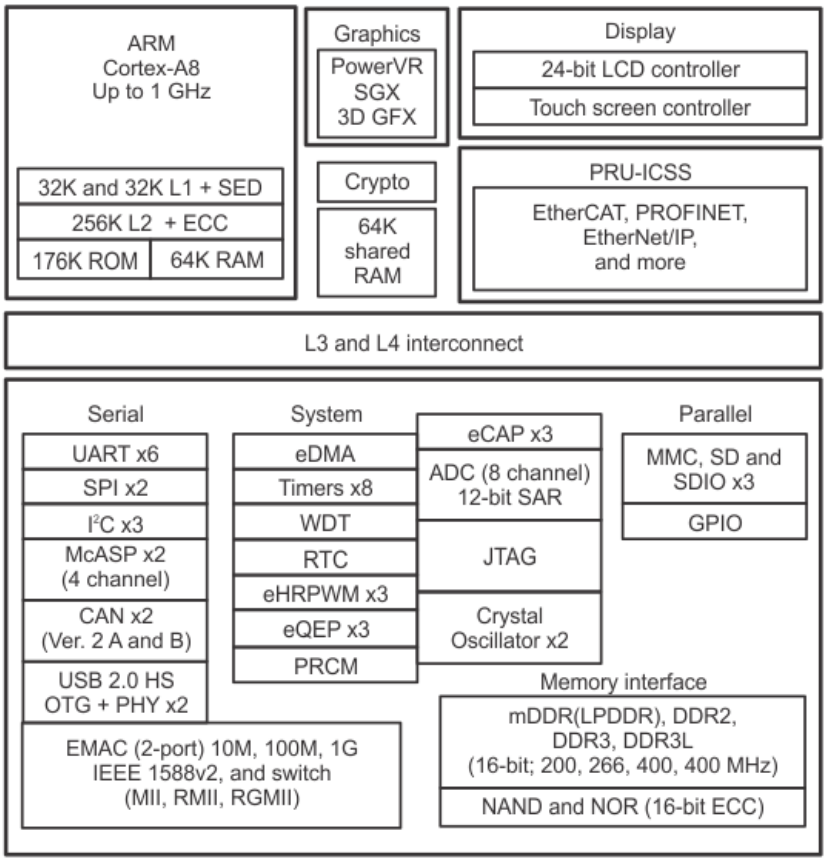

Figure 2.4: AM335x functional block diagram adapted from [3].

The ARM Cortex-A8 CPU in the Sitara Processor operates in a Linux system. There are two images to choose from, Debian and Angstrom. Our project used Debian because it has a wider support. All the data analysis can be performed in this ARM CPU. The problem is that Linux lacks a real-time system to sample data deterministically, and it has a slow I/O, around 10kHz, which is not fast enough for our target data sampling rate. Fortunately, there is a programmable real-time unit (PRU) on the Sitara Processor which can fill in these gaps. The PRU samples data with accurate time stamps, disregards any interrupts from background or elsewhere, and can run at 200 MHz, which is fast enough for our device. The only minor disadvantage of this setup is that the PRU is coded in Assembly, which could make it a little bit harder to perform fancier functionalities. There is 64K shared RAM which both the ARM and PRU can access; thus after the PRU

samples the data at an accurate time stamp, it exports the data to the shared RAM, and then the ARM processor takes the data over for further analysis. Duo CPU also gets the work done faster, which can be critical in the future as we raise the sampling rate further.

For communication interface, the Sitara processor has 6 universal asynchronous receiver/transmitter (UART) interfaces, 2 serial peripheral Interfaces (SPIs), and 3 I^2C interfaces, which leaves plenty of space for choosing peripheral components that use communication interfaces. It has a single 256Mb x16 DDR3L 4Gb memory device on board for temporary data storage when necessary.

For internet, the Beaglebone can use either the Ethernet cable or a USB wi-fi dongle. For battery, Beaglebone has pulled out battery connection pins as TP5 - TP8 for external battery system connections.

Other than having all the required functionalities, what makes Beaglebone Black stand out from other single-board computers is that both the hardware and software of BBB are open-sourced, which means the schematics and PCB layouts are all available if any modification to the hardware needs to be done.

2.2.2 GPS

The Adafruit Ultimate GPS Breakout, shown in Fig. 2.5, was selected.

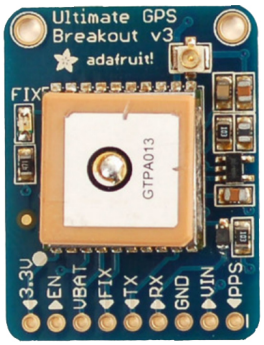

Figure 2.5: Top view of Adafruit Ultimate GPS Breakout.

This GPS board is built around the FGPMMOPA6H MT3339 GPS module. This module is 5 V friendly and ideally has a 20 mA current draw.

This GPS has a UART interface to communicate with the micro-controller (trough pin TX and RX) for time and location information as shown in Fig. 2.6, and also a pulse per second (PPS) output which can be used for time synchronization. Fig. 2.6 shows a series of sample output data from the GPS which are transferred to the micro-controller through UART. The six-digit number right after the "GPRMC" (recommended minimum specific GPS/Transit data) is the "hhmmss" used for time information [18].

```
$GPGGA,202410.000,4042.6000,N,07400.4858,W,1,4,3.14,276.7,M,-34.2,M,,*63
$GPRMC,202410.000,A,4042.6000,N,07400.4858,W,0.08,161.23,160412,,,A*70
$GPGGA,202411.000,4042.5999,N,07400.4854,W,1,3,17.31,275.8,M,-34.2,M,,*5D
$GPRMC,202411.000,A,4042.5999,N,07400.4854,W,0.14,161.23,160412,,,A*7A
```

Figure 2.6: Serial data output of the GPS.

Fig. 2.7 shows the time relationship between the PPS signal and the serial data. The pulse signals the start of a second, then all the data sampled within this second is stamped by the time information acquired through UART. Since the device desires to output 20 data per second, it will rely on the system clock to divide one second into 20 pieces. The resulting time portion of the final output data of the device is shown in Fig. 2.7.

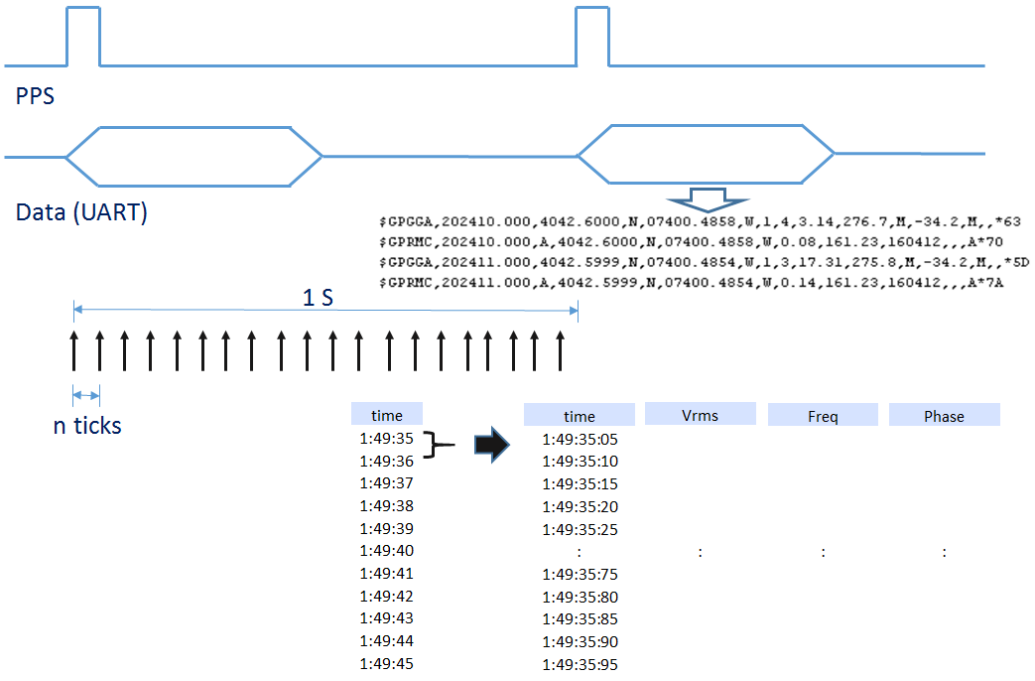

Figure 2.7: Diagram of how GPS data contribute to accurate time-stamps.

Considering that the in-doors usually has very poor satellite reception for

GPS, the Adafruit Ultimate GPS Breakout also has a uFL connector built in for an external active antenna. For the DGAP, an external active antenna with a 5 meter long cable is selected so the antenna can reach out of a window for better satellite reception. There is a "FIX" output that can be connected to an external LED to indicate if the GPS is getting satellite. This signal will be pulled high every second if there is no satellite reception, and will be pulled high every 15 seconds for good satellite reception.

The module can be enabled or disabled easily by pulling the "EN" pin.

2.2.3 Charge Management IC and Li-ion Battery

The backup power supply system is composed of a charge management IC and a battery. This subsystem is connected to Beaglebone Black at pins TP5-TP8, which are presented as a 2x2 header right under the 5 V DC input jack.

By lab observation of DGAP Prototype Version 1.0, the DGAP could draw up to 500 mA current with 5 V DC input but never exceed this amount. A Li-ion battery with features listed in Table 2.1 was selected. The battery can continuously provide 500 mA current to the device and can output 1 A for a short period of time if any unusual power withdrawal occurs.

Table 2.1: Polymer Li-Ion Cell Features [1]

Nominal Capacity	500mAh 1.85Wh
Charging rate	0.5A Max. (1.0C rate)
Continuous Discharging Rate	Continuous: 0.5A (1C)
Max Discharging Rate	1A (2C) for 30 second

Ideally the four pads TP5-TP8 are provided to allow access to the battery pins on the built-in single-chip PMIC TPS65217 which perform the charge and discharge of the battery power; however, during testing, the PMIC was able to discharge the battery to provide power to the system, but was unable to charge the battery when external USB or DC power is present. Thus

charge management IC was needed to perform battery charging. Considering the battery's maxim charging rate is 500 mA, MCP73811, which has a selectable charge current of 85 mA/450 mA [19], was selected for battery charge management. Even though TPS65217 on board is not performing the battery charging alone, the battery charging circuit is still connected to Beaglebone through the four pads which are connected to TPS65217, thus the max output current of the TPS65217 was checked to make sure the charging current does not exceed the PMIC's maxim charging current, which is 1.2 A. For a charging rate of 450 mA, which is the application in our device, the PMIC chip is connected as shown in Fig. 2.8; in the future, if withdrawing 450 mA becomes too much for Beaglebone Black to handle, pin 5 can be grounded to select a charging rate of 85 mA.

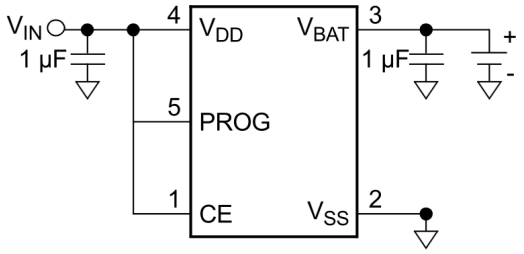

Figure 2.8: Connection of MCP73811 for charging rate of 450 mA.

TP7 on BBB was left floating since the temperature sensor input is not necessary in our case. Note that if the BBB is running off the battery power, the system voltage pins will output 3.3 V instead of 5 V, which is the voltage when BBB is running off the external USB or DC jack.

During prototype testing it was found that if the device is powered from the 5 V DC power jack, when the power is removed, BBB turns off regardless of the present of the battery; but if the device is powered from USB, the BBB keeps functioning with power provided by the battery. Then it was found that, given how the BBB is implemented, if the present 5 V DC power is removed, the power pin to the ARM will be toggled and power down. Thus for the device to smoothly transfer from external power to battery power when needed, the power needs to be provided from USB, but not the 5 V DC power jack.

2.2.4 Linear Regulator

The linear regulator in the device will contribute two parts: the reference voltage of the ADC and the reference voltage V_{ref} in the voltage-step-down circuit. Typically most linear regulators provide output voltage of 1.8 V, 2.5 V, 2.8 V, 2.9 V, 3.0 V, 3.1 V, 3.3 V, 5.0 V, and 10 V to choose from. We want the output voltage of the linear regulator to be as high as possible because a high reference voltage gives the ADC a wider input range, greater value per bit, and thus better accuracy; but since the Beaglebone Black outputs 5 V through the power isolator to the components on the high voltage side, the value of the linear regulator is limited. The linear regulator with 5.0 V could be the best choice if it is guaranteed that the power isolator always provides ≥ 5 V to the V_{in} of the linear regulator, but because there can be disturbance and voltage drop from the power isolator or other cause from the system, the 3.3 V is selected to be safe. Since the DGAP has an accuracy standard to meet, we want the selected chip to have low output tolerance, low dropout, and low noise. After comparisons, LP2985-33DBVR with features shown in Table 2.2 was chosen.

Table 2.2: LP2985-33DBVR Features [2]

Output tolerance (Standard Grade)	1.5%
Dropout rate	280 mV at 150 mA
	7 mV at 1 mA
Noise (with 10-nF Bypass Capacitor)	$30\mu V_{RMS}$

LP2985-33DBVR has a relatively low output tolerance, low dropout, and low noise compared to a selection of other linear regulators and requires fewer external components. The external capacitors are connected as shown in Fig. 2.9 for low-noise operation and output stability. The enable pin (pin 5) is directly connected to high since the output of the linear regulator is constantly needed.

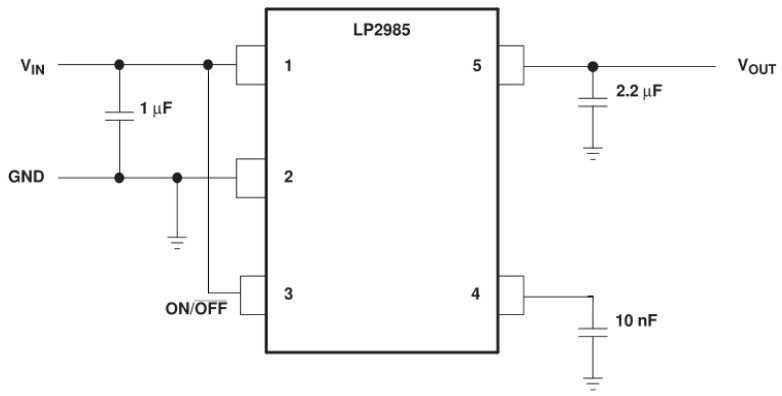

Figure 2.9: Connection of LP2985-33DBVR.

2.2.5 Analog to Digital Converter

The analog-to-digital converter (ADC) takes in a voltage reading between 0 V and the V_{ref} which is 3 V, and outputs a series of digital signals which can be read by the micro-controller. The analog value for the least significant bit (LSB) of the serial of digital signal can be calculated as

$$LSB = \frac{V_{ref}}{2^{\#bits} - 1} \tag{2.3}$$

For example, there is a 12-bit built-in ADC in the Sitara processor, so if we use a reference voltage of 3.3 V, the value of the LSB of this particular ADC is

$$LSB = \frac{3.3\ V}{2^{12} - 1} \simeq 0.8059\ mV \tag{2.4}$$

Therefore, if the micro-controller reads 2310 from this ADC, for example, the voltage at the input of the ADC can be calculated as

$$V = 0.8059\ mV \times 2310 = 1.8615\ V \tag{2.5}$$

The value of the LSB is one factor that determines how accurate the ADC reading can be. An ADC with LSB = 0.8059 mV means the voltage read can

only increment by 0.8059 mV, thus it will not reflect the voltage accurately if the voltage variation is less than 0.8059 mV. Decreasing the value of the LSB increases the accuracy of ADC reading. Also notice that the voltage the ADC reads is scaled down from $\pm 185V$ to 0.1 V to 3.2 V, thus the error will be even larger after the 1.8615 V is transformed back to the full scale. From this aspect, the 12-bit ADC on Beaglebone Black could be too inaccurate to be used in the DGAP to meet the IEEE standard [20].

A 14-bit ADC ADC141S626 was used in the device. As ADC141S626 takes both positive and negative voltage inputs, which is very different from many other ADCs, and its most significant bit (MSB) presents the sign of the voltage input [21], it is thus really a 13 bit ADC with input voltage that is twice wider. Prototype Version 2.0 only used the positive part of the input, which can be improved in the future with some modification of the voltage-scale-down circuit. For Prototype Version 2.0, the value of the LSB of ADC141S626 is

$$LSB = \frac{3.3\ V}{2^{13} - 1} \simeq 0.4029\ mV \tag{2.6}$$

The voltage at the power outlet (V_1) and the voltage at the input of the ADC (V_2) can be described by a linear equation $V_1 = aV_2 + b$. To scale back the voltage reading, we know

$$\begin{cases} -185 = 0.1a + b \\ 185 = 3.2a + b \end{cases} \tag{2.7}$$

Solving Equation 2.7, we get $a \simeq 119.36, b \simeq -196.94$, thus

$$V_1 = 119.36V_2 - 196.94 \tag{2.8}$$

which means the measured voltage value increments by

$$LSB = 119.36 \times 0.4029\ mV = 0.048\ V \tag{2.9}$$

This value will be halved if the negative portion of the input range is also used. The real value could be off by a little bit due to the tolerances of the components such as the resistors in the voltage-scale-down circuit, the linear regulator output voltage, etc., thus calibration is needed before put the device in field.

ADC141S626 has a maxim sampling rate of 250 kSPS, which is more than enough to provide our desired 10 kSPS sampling rate. It uses the SPI communication interface which is supported by Beaglebone Black. It takes the system clock (SCLK) and chip selection (CS) signals, which determine the sampling rate, from Beaglebone Black and feeds back sampled data D_{out}. The external capacitors are connected as shown in Fig. 2.10 for low-noise operation and input stability.

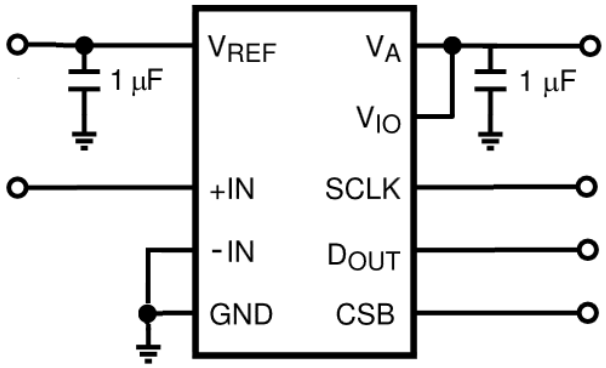

Figure 2.10: Connection of ADC141S626.

2.2.6 Digital & Power Isolators

Digital and power isolators are used between BBB and all the components on the high voltage side, including the ADC and the linear regulator, to isolate any signal and power route. This measure can be very beneficial from the safety aspect; it protects the system on the low voltage side (including the Beaglebone Black, the GPS, the backup battery system) from abnormal current or voltage caused by fault in the power grid or broken parts in the high voltage side which may damage components in the low voltage part or cause safety hazards.

There are many isolators to choose from. For the DGAP, there are 3 signals to be isolated between BBB and the ADC: from BBB to ADC, we have system clock and the chip selection signal; from ADC to the BBB, there is the sampled data. And the BBB needs to provide power to ADC and the linear regulator through a power isolator. The ADuM640x series stands out since it includes the signal isolation channels and power isolator in one single chip. The power isolator can take 5 V or 3.3 V input from the BBB and output 5 V or 3.3 V to the high voltage side, which means even if the BBB is running off the battery power, the system on the high voltage side can still have 5V power rail. The power isolator can output up to 400 mW, which is capable of supporting the high voltage side. As a signal isolator, ADuM640x has four channels; the difference between ADuM6400, ADuM6401, ADuM6402, ADuM6403, and ADuM6404 is the direction of the channels, as shown in Fig. 2.11.

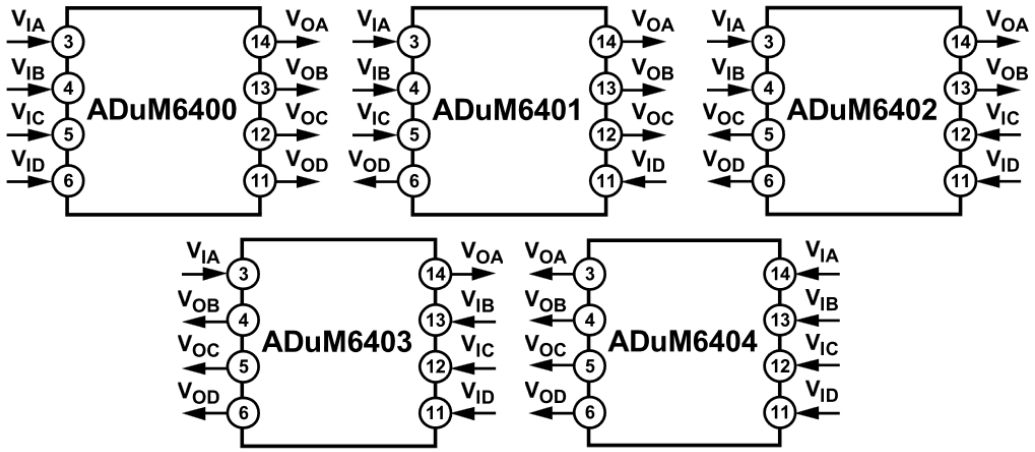

Figure 2.11: Channel directions of ADuM640x series adapted from [4].

The DGAP requires two signal channels in the same direction as the power channel, and one signal channel in the opposite direction; thus both ADuM6401 and ADuM6402 would work, and ADuM6401 was chosen. The pin configuration of ADuM6401 is shown in Fig. 2.12. V_{DDL} (pin 7) is connected to V_{DD1} (pin 1) to supply power to the data channels, and V_{SEL} (pin 10) is connected to V_{ISO} (pin 16) to select 5 V output voltage ($V_{ISO} = 5V$).

From the data sheet, it was noted that 5 V output with a 3.3 V input is discouraged since this combination has quite low power efficiency. But since power consumption is not a critical consideration in this project at this stage,

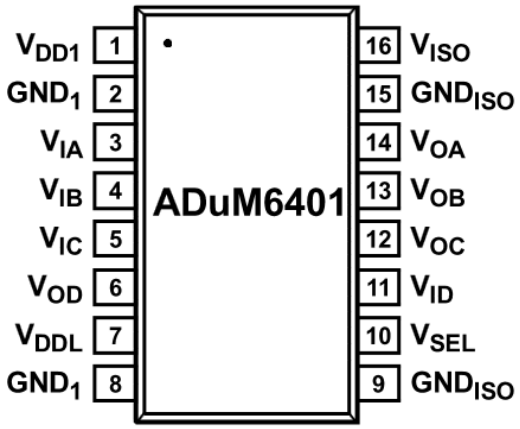

Figure 2.12: Pin configuration of ADuM6401.

Prototype Version 2.0 is fine with this setting.

2.2.7 Resistors in the Voltage-Step-Down Circuit

In Section 2.1, we have discussed the voltage-step-down circuit and obtained the ratio between R_1, R_2, and R_3 from Equation 2.2 to be 58:1:1. When choosing the value of the resistors, we need to consider both the power rating and the noise. The resistance value can be large to prevent reaching the power ratings, but as the resistance value increases, the noise at the output will increase as well; since data accuracy is critical to a DGAP, we want to keep the resistor values as low as possible.

Most surface mount resistors with unit $k\omega$ usually have power rating of 1/8 W or higher. According to Fig. 2.2, the maxim voltage that can be applied to R_2 or R_3 is 3.3 V (disregard the 0.1 V margin). The relationship between resistance R, voltage V, and power P of a resistor is

$$P = \frac{V^2}{R} \tag{2.10}$$

Thus the minimum resistance capable for R_2 and R_3 is

$$R_{2,3} = \frac{(3.3\ V)^2}{1/8\ W} = 87.12\ \Omega \tag{2.11}$$

The maxim voltage that can be applied to R_1 is -185 V (disregard the 0.1 V margin), thus the minimum resistance capable for R_1 is

$$R_1 = \frac{(-185V)^2}{1/8W} = 273.8k\Omega \tag{2.12}$$

To keep the ratio of 58:1:1, R_1 limits the minimum value of R_2 and R_3. $R_1 = 357k\Omega$ was chosen, thus R_2 and R_3 supposed to be

$$R_{2,3} = \frac{357k\Omega}{58} = 6.16k\Omega \tag{2.13}$$

To ensure the measurements to be as accurate as possible, resistors with 1% tolerance are desired, and these have a set of standard values. $6.19k\Omega$ is the closet one to the calculated $6.16k\Omega$ [22], thus it is selected.

Most of the resistors on board, including R_2 and R_3, have the 0603 package. For R_1, referring to Appendix B, we can see that the space between the two pins of R_1 is the only blank between the power outlet and the rest of the circuits behind R_1; thus 1206, a larger package for R_1 which gives a larger space between its two pins, is selected for safety reasons.

2.2.8 Circuit Protection and Signal Indication

As mentioned in Section 2.2.7, the space between the two pins of R_1 is the only gap between the power outlet and the rest of the circuits. Even though we have increased the package size of R_1, more safety measures can be applied. A fuse (250 V /500 mA rating [23]) is inserted between R_1 and the power outlet connector to prevent over current /voltage.

For signal transmission, all the signal routes have a 5Ω series resistor to prevent short circuit. Exceptions include R_{12} and R_{13} which are in series

with the LEDs, indicating the presence of Beaglebone's system voltage (V_{sys}) and 5 V external voltage. Since the LEDs indicate power and are directly connected to the power rails without current limitation, R_{12} and R_{13} need to be larger to limit the current through the LEDs so it does not go over the limit, which is 30 mA for the LEDs selected [24]. By Ohm's law, R_{12} and R_{13} need to be at least

$$R_{12,13} = \frac{5\ V}{30\ mA} = 166.67\ \Omega \tag{2.14}$$

assuming the LED does not have a resistance. 200Ω resistors are selected for R_{12} and R_{13}.

2.2.9 Container

A Hammond Project Box with size that will fit the board was chosen. The box has eight stands to screw in the PCB. Holes are hollowed out at the front and side of the box as shown in Fig. 2.13 for the connectors and LEDs.

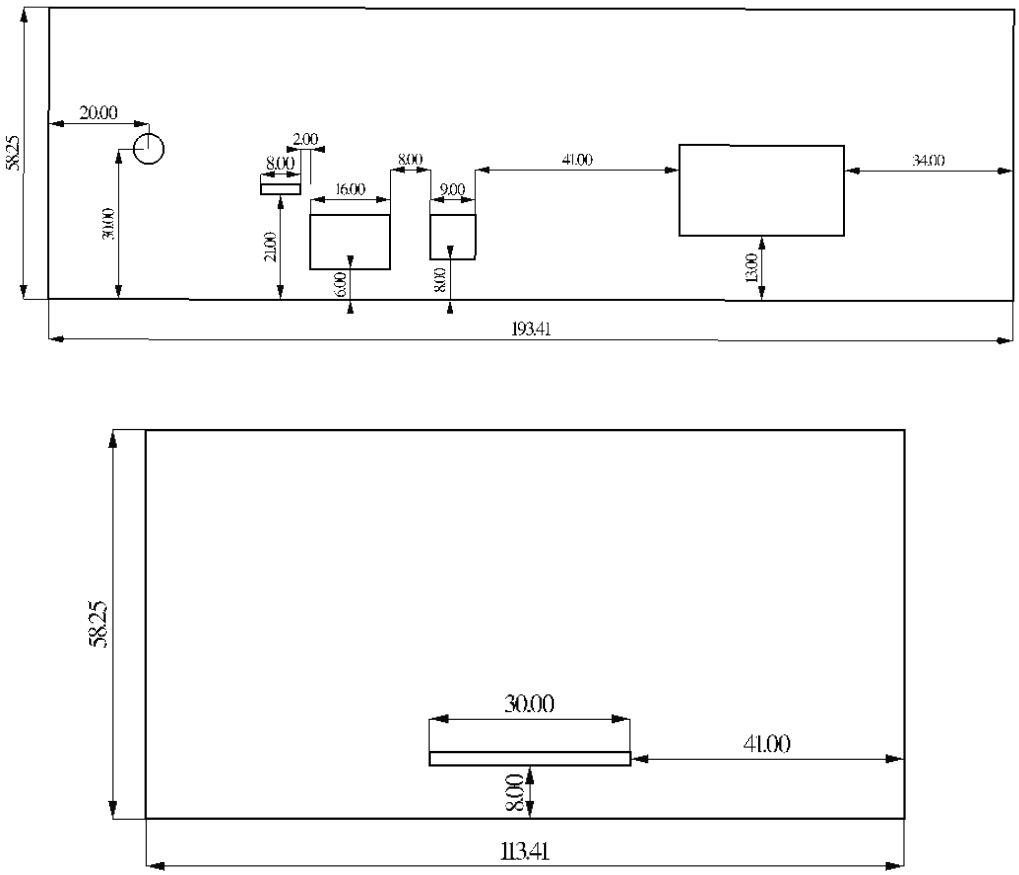

Figure 2.13: Regions hollowed out on the box for connectors and LEDs. Front view (top) side view (bottom).

Refer to the box data sheet [25] for dimensions of the box.

2.3 Prototypes

Fig. 2.14 shows the end product. More photos showing the device from multiple views can be found in Appendix C.

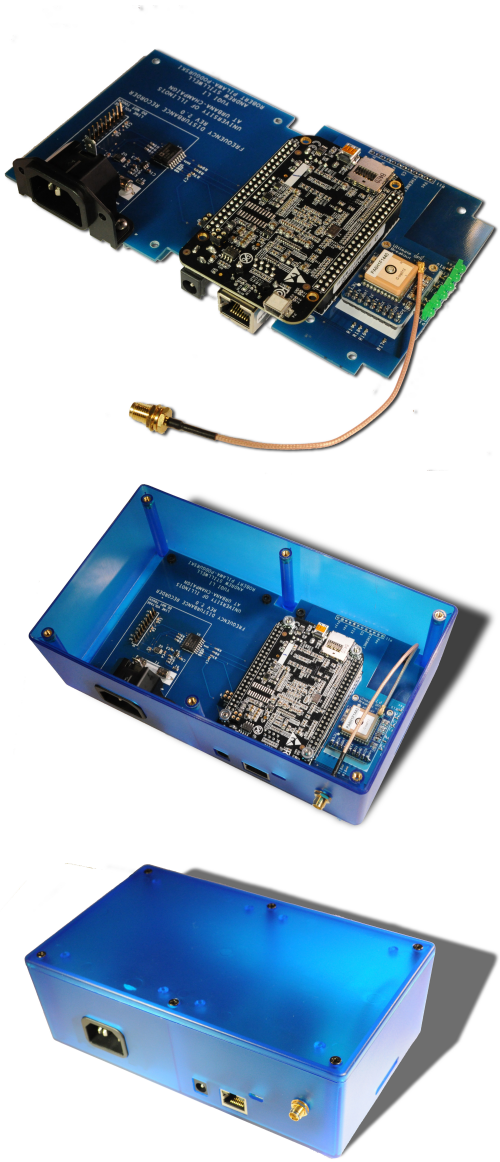

Figure 2.14: Distributed Grid Analytics Platform Rev. 2.0.

Compared to DGAP Rev. 1.0 as shown in Fig. 2.15, Rev. 2.0 has more space between the high voltage side and the low voltage side for safety reasons.

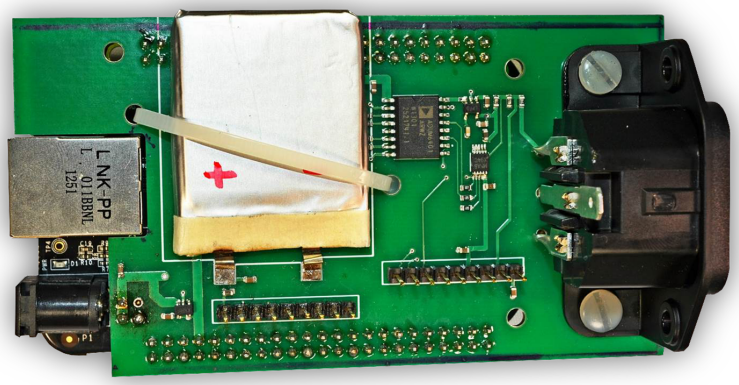

Figure 2.15: Distributed Grid Analytics Platform Rev. 1.0.

Rev. 2.0 has pulled out Beaglebone system voltage (V_{sys}), 5 V DC voltage (V_{DC}), GPS FIX signal (FIX), ADC chip selection (CS), Ethernet enable (Ethernet), and signal that indicates the presence of the power outlet AC voltage (V_{ac}) to six LEDs on the side edge of the PCB. V_{sys} indicates that the device is powered. V_{DC} indicates that the device is drawing power from the external 5 V DC power jack. FIX indicates the GPS satellite reception status, the same pattern as explained in Section 2.2.2. CS indicates that BBB is normally asking ADC to sample data. V_{ac} indicates that the device is finely plugged into the power outlet.

In the high voltage region of the board, there is a row of headers for testing purposes. Signals pulled out include the 3.3 V reference voltage V_{ref}, positive signal input of ADC (+IN) and negative signal input of ADC (-IN), system voltage (VDD) and Ground (GND) of the high voltage side, chip selection of ADC (CS), serial data output of ADC (DATA), and system clock of BBB (CLK).

At the very middle of the board, there is a 2x2 header (P6). P6 is the battery pads corresponding to TP5-TP8 on Beaglebone black. When assembling, after the BBB is secured on the PCB, use wires to connect P6 to TP5 – TP8 on BBB, upper left pin to upper left pin, lower right pin to lower right pin, respectively.

Use screws to secure the Beaglebone Black to the PCB, power outlet con-

nector to PCB, PCB to the box, and container lid to the container base.

CHAPTER 3

SOFTWARE IMPLEMENTATION

3.1 Software Work-flow

The system level software work-flow for the DGAP is shown in Fig. 3.1

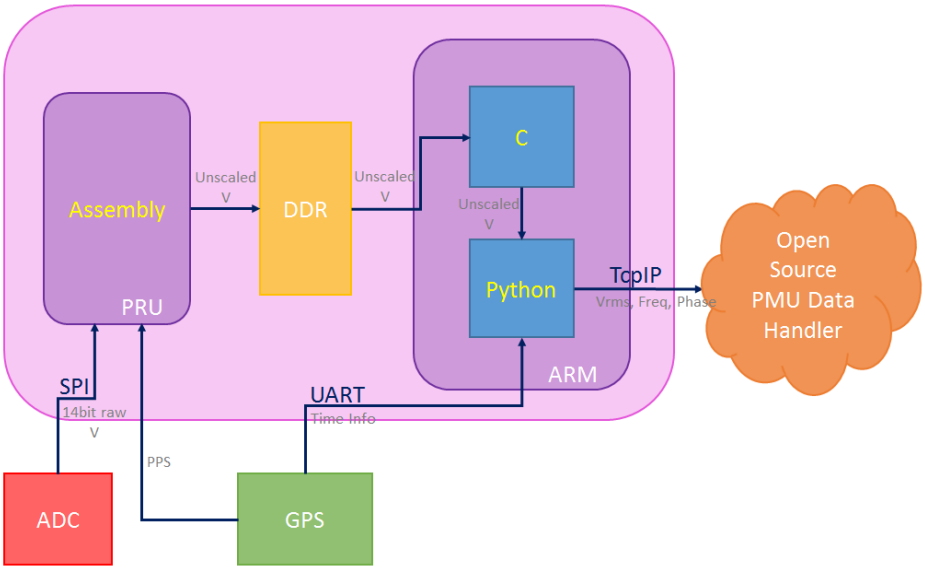

Figure 3.1: Software work-flow of the FDR.

The general idea is as follows. One of the programmable real-time units (PRUs) on board acquires 14 bits raw data from the ADC and transfers it to the shared memory; the other PRU counts the number of micro-controller timebase ticks between pulse per second (PPS) signals from GPS, which will be used for time synchronization. On the other side of the shared memory, the ARM processor will use C code to draw the unscaled raw data and transfer it to Python for further processing. The GPS also provides time information to ARM through UART. After the raw data get scaled and the frequency,

V_{rms}, and phase are calculated, the data in defined chunk sizes are sent to an open source PMU data handler called Open PDC through TCP/IP. The Assembly, C and Python code are given in Appendix F.

3.2 Mathematical Methods for RMS Voltage, Frequency, and Phase Calculation

After the voltage data and the time stamps are all imported to the Python code, different mathematical methods are compared to choose the one that gives the most accurate RMS voltage, frequency, and phase measurements. This section will present the mathematical methods used for data calculation utilized by DGAP.

3.2.1 RMS Voltage Calculation

By definition, the root mean square (RMS) value of a set of values is defined as the square root of the arithmetic mean of the squares of the values as shown in Equation 3.1.

$$x_{rms} = \sqrt{\frac{1}{n}(x_1^2 + x_2^2 + \cdots + x_n^2)} \tag{3.1}$$

The DGAP calculates V_{rms} in the exact same way. The device calculates V_{rms} every 50 ms, with the sampling rate of 20 kSPS; the equation for V_{rms} calculation is

$$V_{rms} = \sqrt{\frac{1}{1000}(V_1^2 + V_2^2 + \cdots + V_{1000}^2)} \tag{3.2}$$

which is implemented into the Python code.

3.2.2 Frequency Calculation

Frequency is one of the most important parameters of a power system: load unbalance is always faithfully portrayed by a change in the operating frequency [26] [27]; thus the obtained frequency data need to be as accurate and as transient as possible. The DGAP calculates frequency using fast Fourier transform (FFT). Fast Fourier transform is the Fourier transform of a block of time data points and represents the frequency composition of the time signal. An example of a 20 Hz perfect sine wave and its FFT is shown in Fig. 3.2.

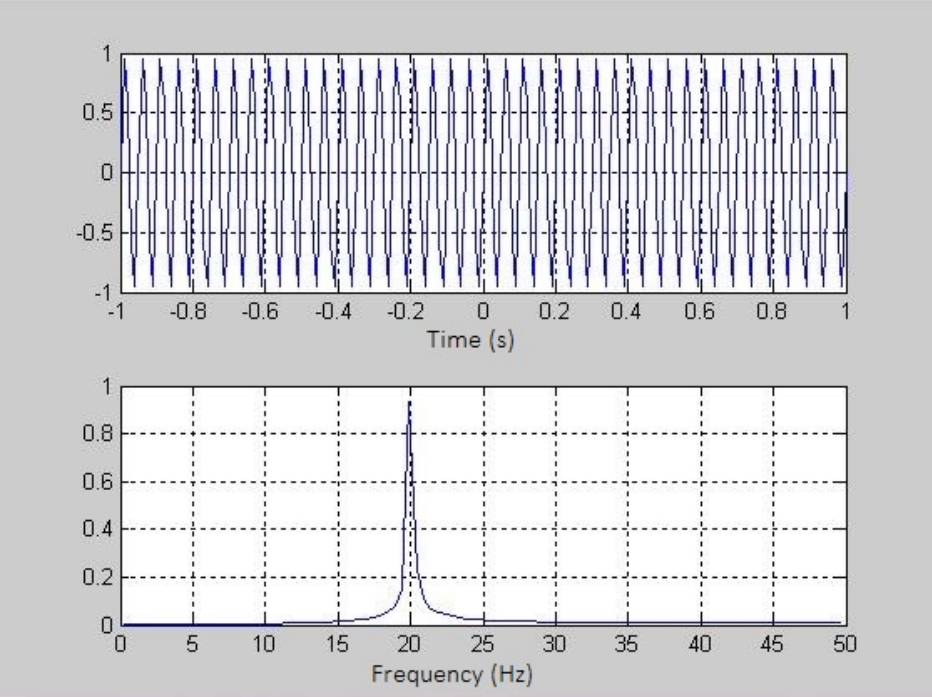

Figure 3.2: Time waveform of sine function (top) and its FFT (bottom).

The sine wave in Fig. 3.2 is indicated by a single discrete peak in the FFT with height of 1.0 at 20 Hz. This is the case when the sine wave is perfect and there are integer number of cycles within the window. If there is not an integer number of cycles in the window, there will be leakages resulting in signal energy leaking over a wide frequency range in the FFT instead of a narrow frequency range as shown in Fig. 3.3 [28].

To reduce this smearing effect, a window is applied to the data before taking FFT. A window is shaped so that it is exactly zero at the beginning

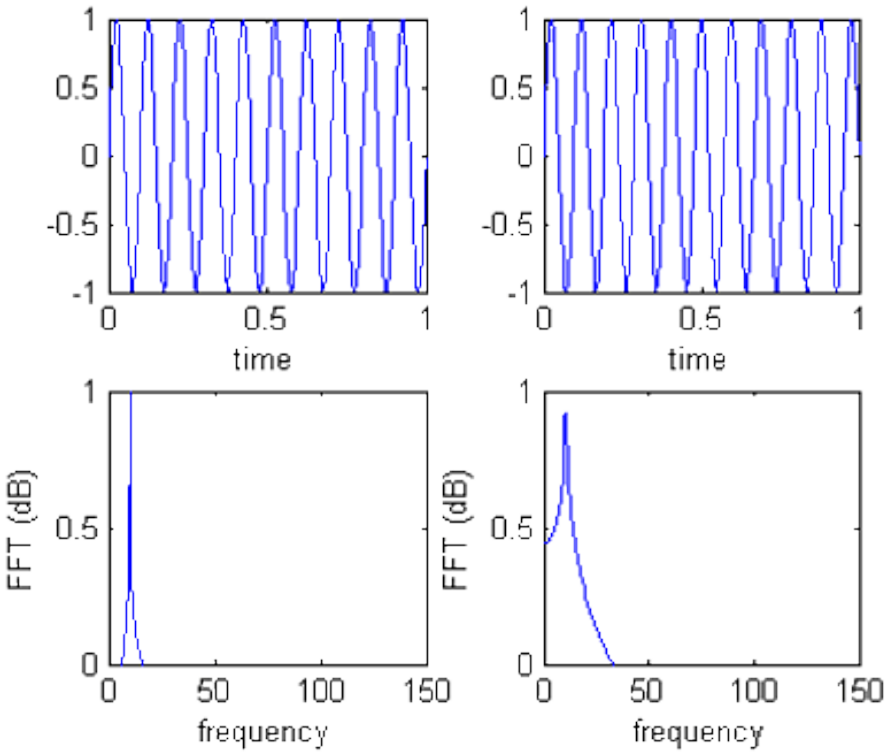

Figure 3.3: FFT of since wave with integer number of cycles (left) and non-integer number of cycles (right).

and end of the data block, which reduces the contribution of the edges to the FFT spectrum and minimizes the effect of leakage to better represent the frequency spectrum of the data. There are several windows to choose from, each with advantages or disadvantages of frequency resolution, spectral leakage, amplitude accuracy, and so on, and best for different signal types. After trying out different windows on data collected by DGAP, the Hanning window was chosen.

With the windowed FFT, a parabolic fit was applied to the data to find the peak more accurately.

3.2.3 Phase Calculation

The phase was simply returned as an angle parameter by the Python rfft function.

CHAPTER 4

EXPERIMENTAL RESULTS

4.1 Testing Setup

The experimental results are taken with the hardware prototype described in Chapter 2 and software setups described in Chapter 3. The bench setup for the experiment is shown in Fig. 4.1

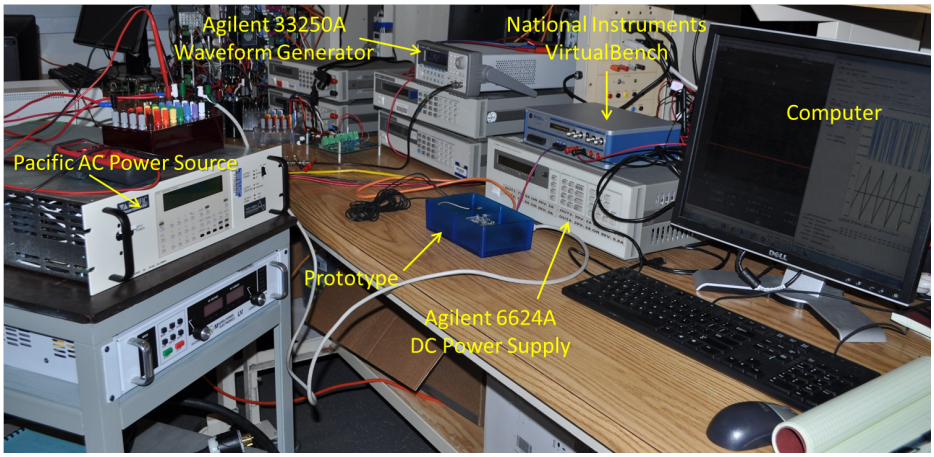

Figure 4.1: Bench setup for DGAP testing.

First, the prototype needs to be calibrated due to the tolerances of components such as resistors in the voltage-scale-down circuit, the linear regulator output voltage, etc., as explained in Chapter 2.

Referring to Section 2.2.5, we need two points to get the linear relationship between the raw data read and the actual voltage value. An Agilent 6624A System DC Power Supply that can provide up to 50 V was used to provide stable DC voltage to the measuring terminal of the prototype. Two voltages, 0.3 mV and 50.016 V, were measured with readings of 3993.0 and 5061.8

(average of several measurements for more accurate results). The relationship between raw data and voltage is

$$\begin{cases} 0.0003 = 5061.8a + b \\ 50.016 = 3993.0a + b \end{cases} \tag{4.1}$$

Solving Equation 4.1, we get a = 0.0468 and b = -186.86; thus

$$V_{actual} = 0.0468V_{raw} - 186.86 \tag{4.2}$$

Equation 4.2 was updated to the Python code in Appendix F.3.

The DGAP is powered by a 5 V DC supply through a National Instruments VirtualBench.

Before testing the DGAP with some dangerous voltage such as 120 V AC, we tested it with an Agilent 33250A 80MHz Function/Arbitrary Waveform Generator, which can generate a sinusoidal waveform with amplitude up to 20 V peak-to-peak. Agilent 33250A is safer since it limits the current output and thus will not damage the board if anything goes wrong. After a simple test by Agilent 33250A, a Pacific AC Power Source, which can provide up to 400 V two-phase power, was used to pretend the power outlet. We want to test the prototype by a power source instead of plugging it directly to the wall outlet because with a power source with controllable voltage amplitude and voltage frequency, we can see how accurate the measurements are and how the measurements behave as the voltage and frequency go higher or lower than the nominated 120 V_{rms} 60 Hz.

4.2 Test with Controllable AC Power Source

4.2.1 Functionality Test

PMU Connection Tester was used to monitor the real-time data collected. The nominated voltage of the power grid – 120 V_{rms} (power source have actually provided 120.3 V_{rms}), 60.00 Hz, was tested first, and the results are presented in the PMU Connection Tester as shown in Fig. 4.2

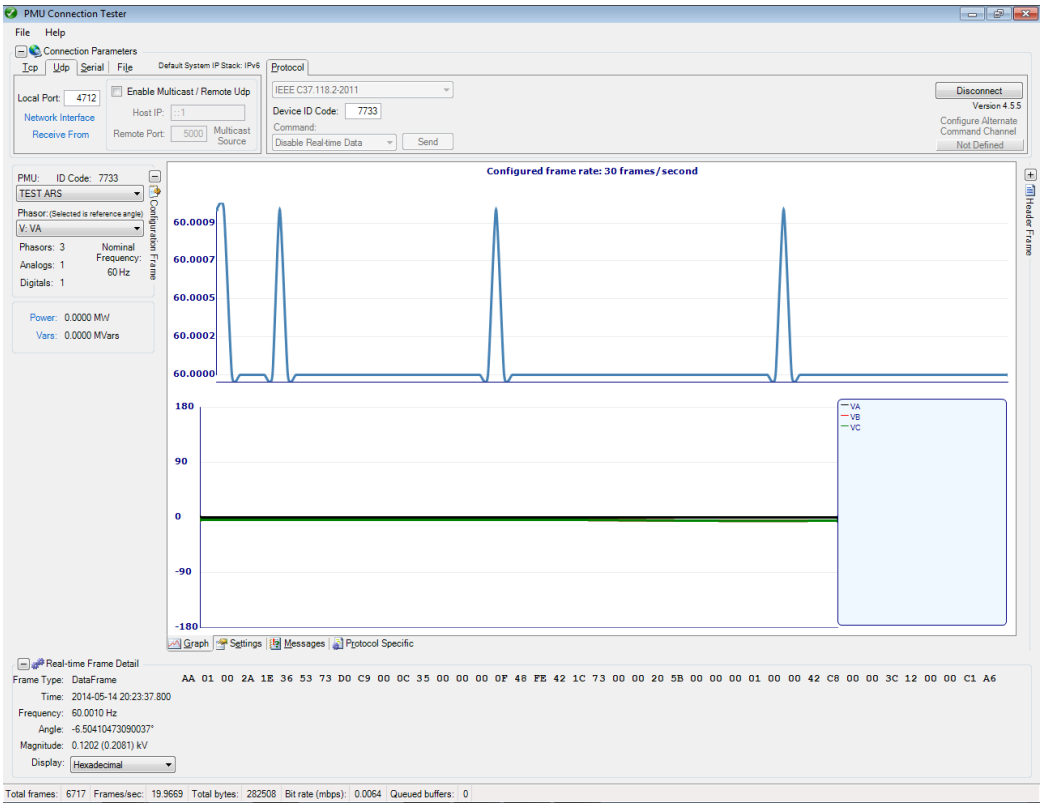

Figure 4.2: PMU Connection Tester workspace (120 V_{rms} 60 Hz).

The upper plot shows the frequency reading and the lower plot shows the phase reading. The frequency readings are quite accurate at 60.0000 Hz with some disturbance to 60.0009, which is within the tolerance. The phase readings are kept at 0 degree (depending on the initial phase measurement), which is expected since we are taking one measurement every 50 ms. Within 50 ms there are three full cycles, so we expect to have the same phase measurement every time. This is not the case for frequency higher or lower than

60 Hz since the cycles are not complete within 50 ms; the positions at which we measure phase for the sinusoidal waveform are different (shifting) every time we sample. An example with 120 V_{rms} 59.00 Hz is shown in Fig. 4.3

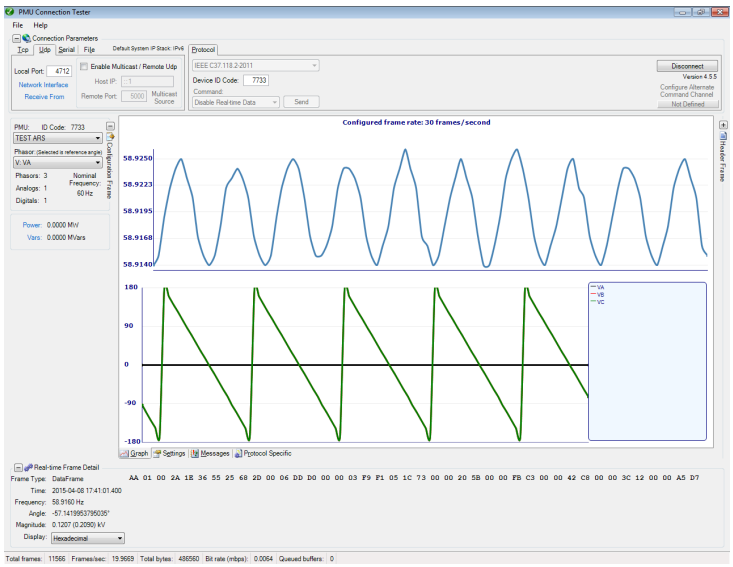

Figure 4.3: PMU Connection Tester workspace (120 V_{rms} 59 Hz).

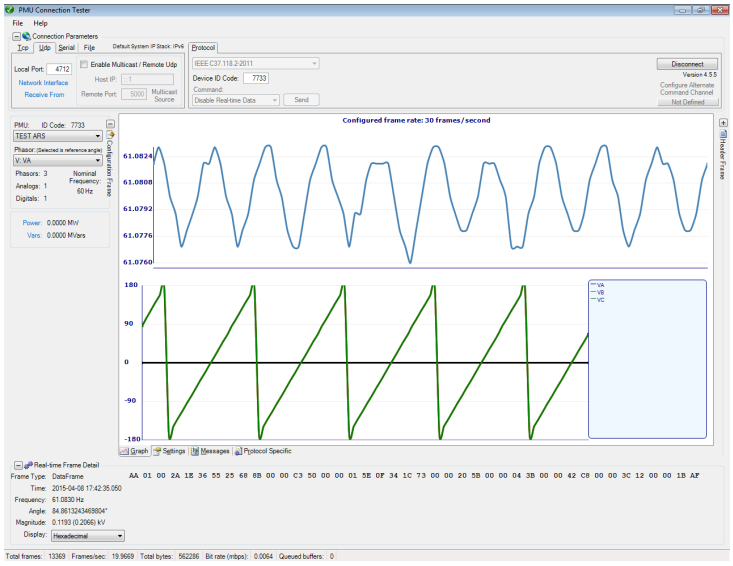

Figure 4.4: PMU Connection Tester workspace (120 V_{rms} 61 Hz).

One can see that the phase shifts between -180 degrees and 180 degrees. Compare to Fig. 4.4, we can see that the phases changes with slope: for 59 Hz the phase increases as time goes on, for 61 Hz the phase decreases instead, behaviors that are expected.

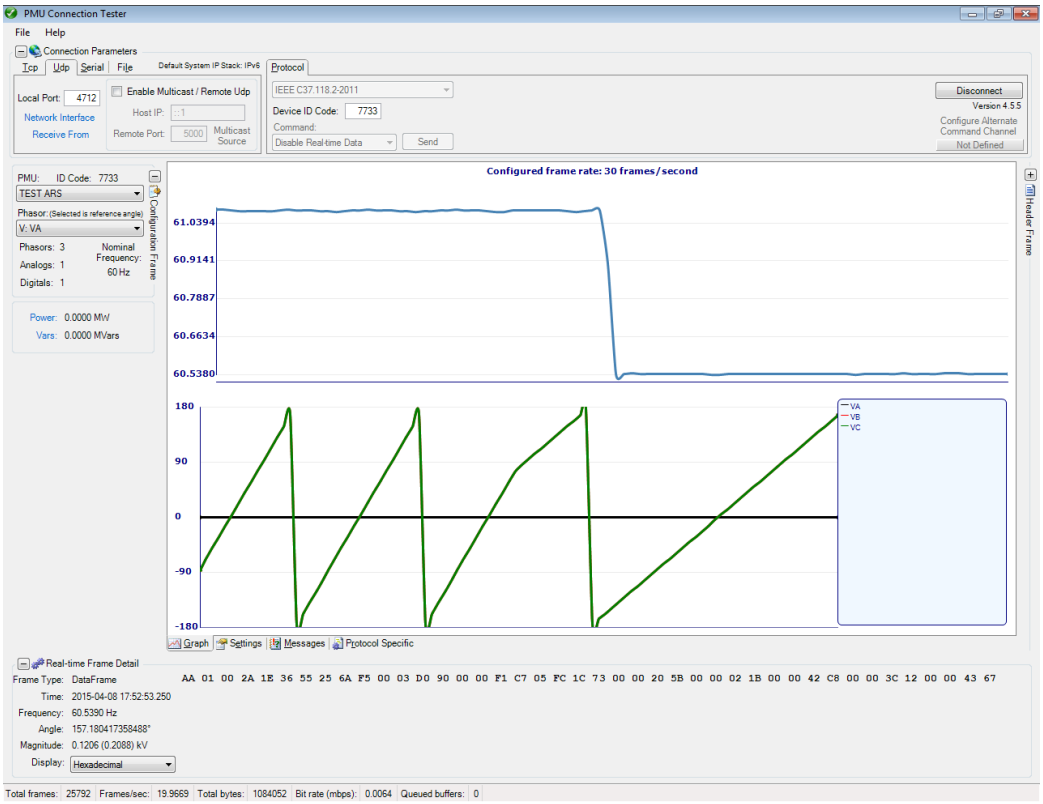

Figure 4.5: PMU Connection Tester workspace (120 V_{rms}, 61 Hz to 60.5 Hz).

Also, as the frequency get further away from 60 Hz, the phase shifts faster as the cycle screw will increase (cycles are not completed/more data from next cycle include) for every 50 ms. Fig. 4.5 shows the plots when the frequency changes from 61 Hz to 60.5 Hz. The phase shifts much slower for 60.5 Hz compare to 61 Hz.

The PMU Connection Tester shows the RMS voltage in the Real-time Frame Detail window. As the results show in Fig. 4.2 – 4.5, the RMS voltages are all very close to 120 V, with little offset possibly caused by disturbances.

4.2.2 Reliability Test

To test the reliability of the DGAP for input higher or lower than the nominated 120 V_{rms} 60 Hz waveform, frequencies in the range of 59.0 Hz - 61.0 Hz

(actual frequency provided by power supply) are measured and the output (frequency read) is listed in Appendix E, Table E.1. The percentage error is plotted against frequency measured, as shown in Fig. 4.6.

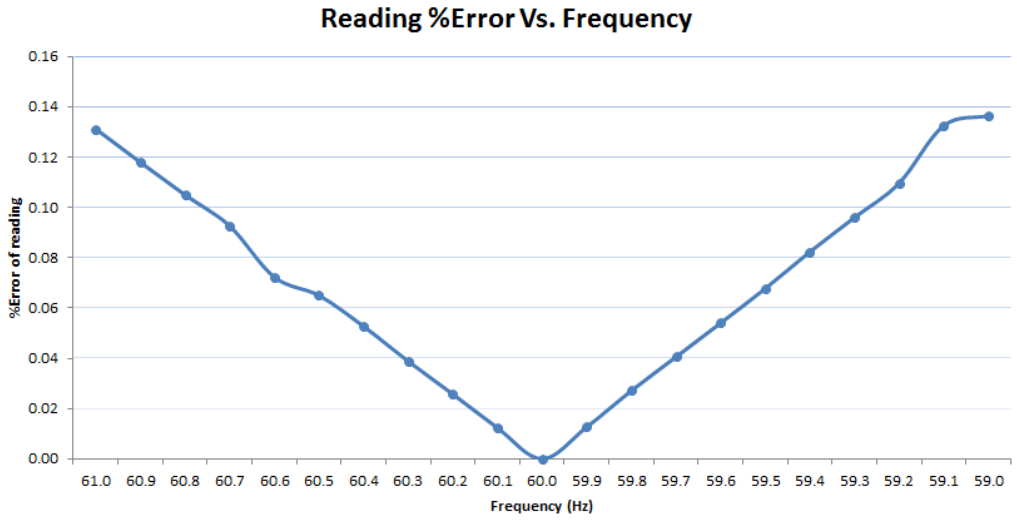

Figure 4.6: Percentage error vs. frequency.

As plotted, percentage error is kept low, but increases as the frequency gets away from 60 Hz. This error may be due to spectrum leakage explained in Chapter 3.

The V_{rms} voltage measured as shown in Appendix E, Table E.2, kept quite close to the output value from the power source, which satisfies the goal and leaves less to discuss.

4.3 Power Outlet Measurements

With the DGAP tested and verified with the AC power source, the device is plugged into the wall to read some real power grid data, which is set to 120 V_{rms} 60 Hz. The data is sent to PMU Connection tester and the result is plotted in Fig. 4.7

The frequency reading is around 60.0050, and V_{rms} is around 117.1 V, which is very close to our expectation. The phase angle is shifting around a little bit since the frequency is not perfectly aligned at 60 Hz. From the plot

it can be seen that the frequency wave is more irregular than the previous waves that get clean waveforms from an AC power source.

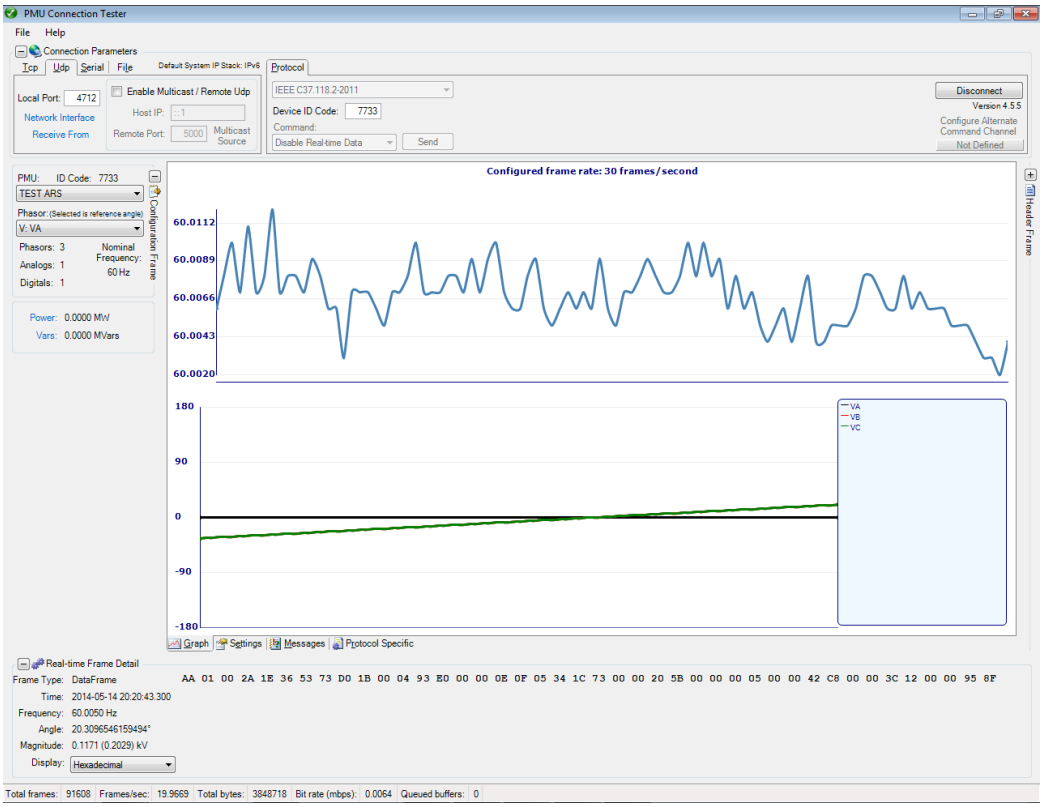

Figure 4.7: PMU Connection Tester workspace (wall outlet).

APPENDIX A

SCHEMATIC DRAWING FOR DGAP REV 2.0

To improve visibility, the schematic drawing of the DGAP is split into low voltage region and high voltage region, which are shown separately in Figs. A.1 and A.2

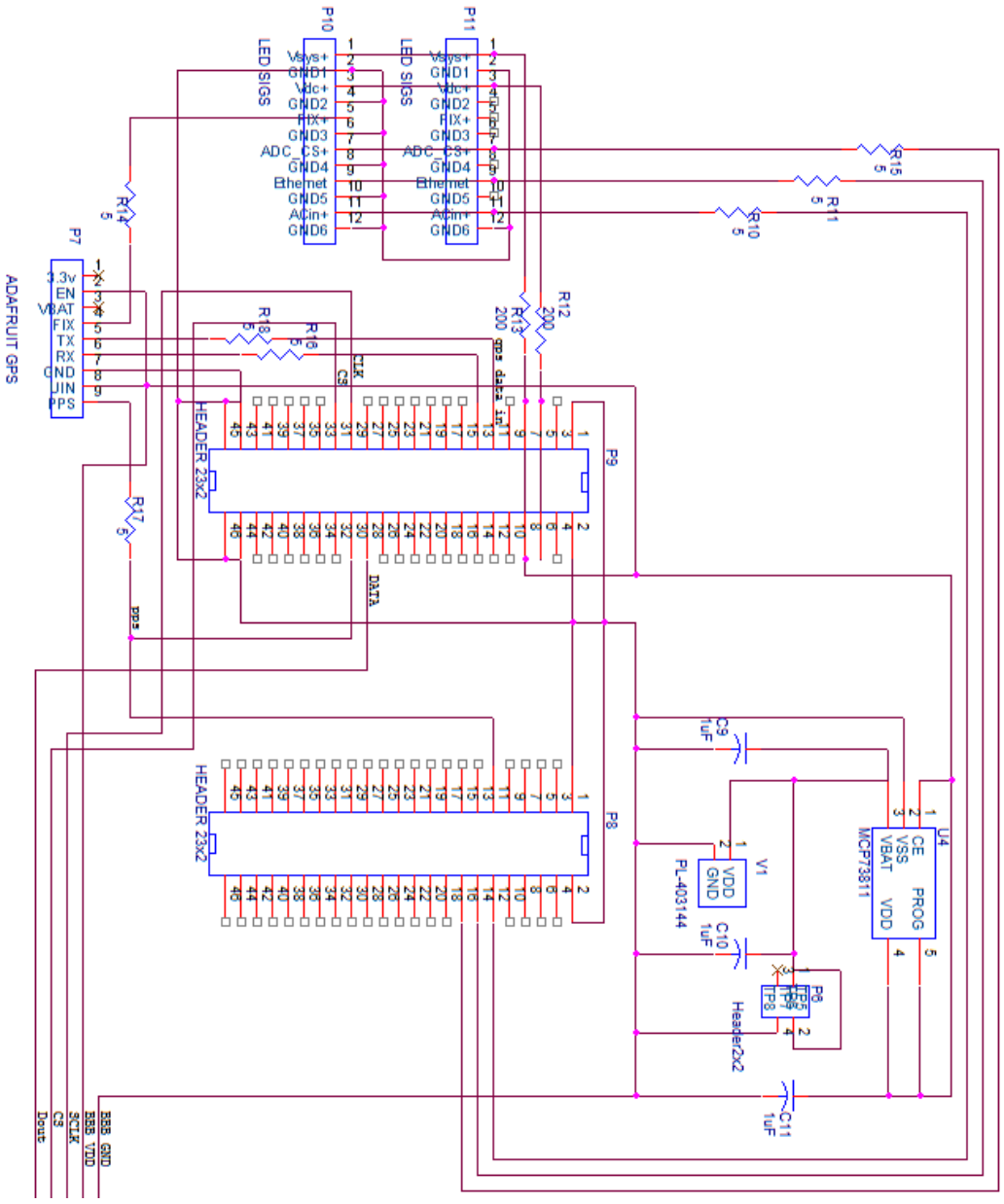

Figure A.1: Low voltage region of the DGAP.

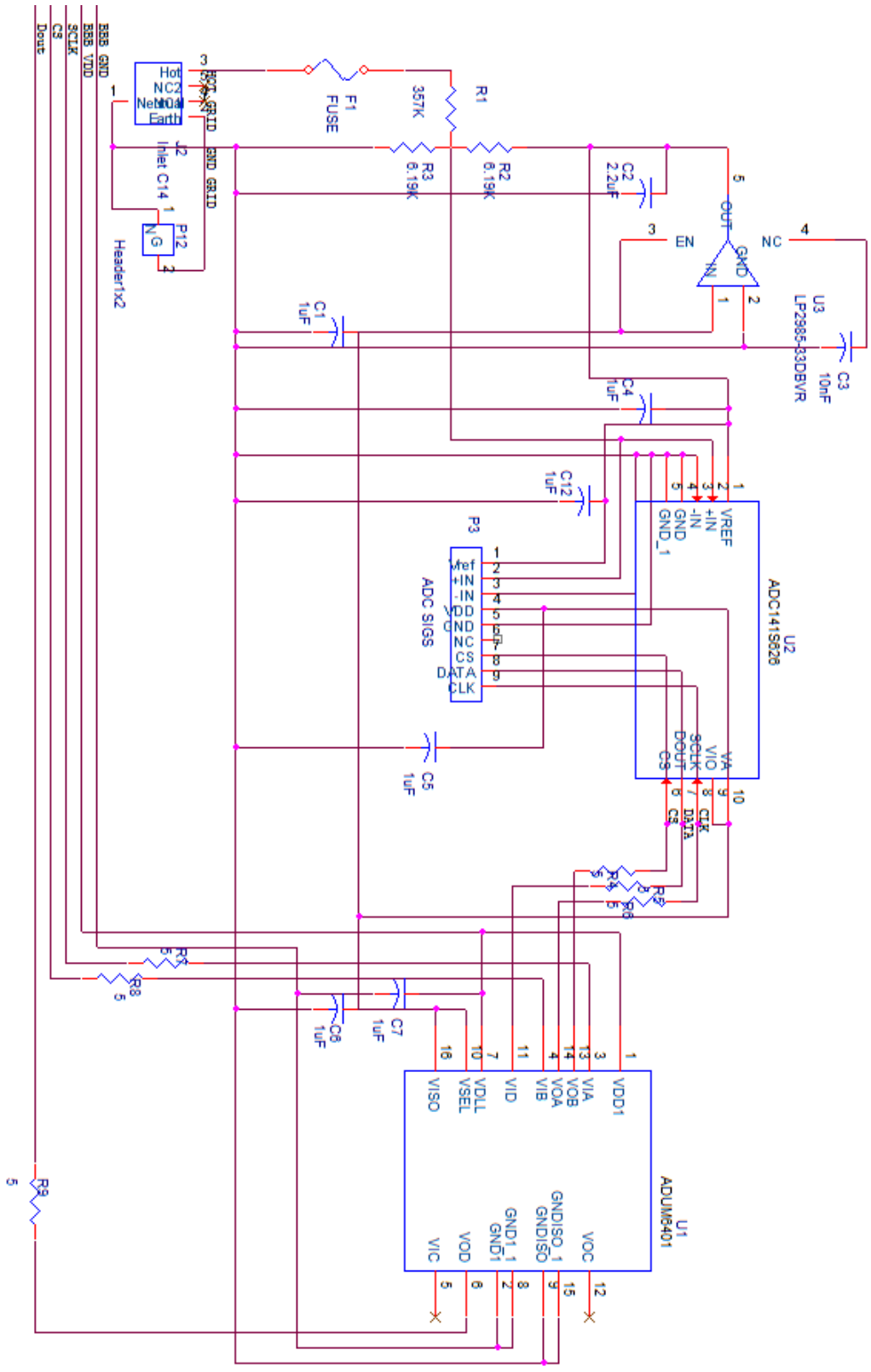

Figure A.2: High voltage region of the DGAP.

APPENDIX B

PCB LAYOUT FOR DGAP REV 2.0

The PCB layout is shown in Figs. B.1 – B.5

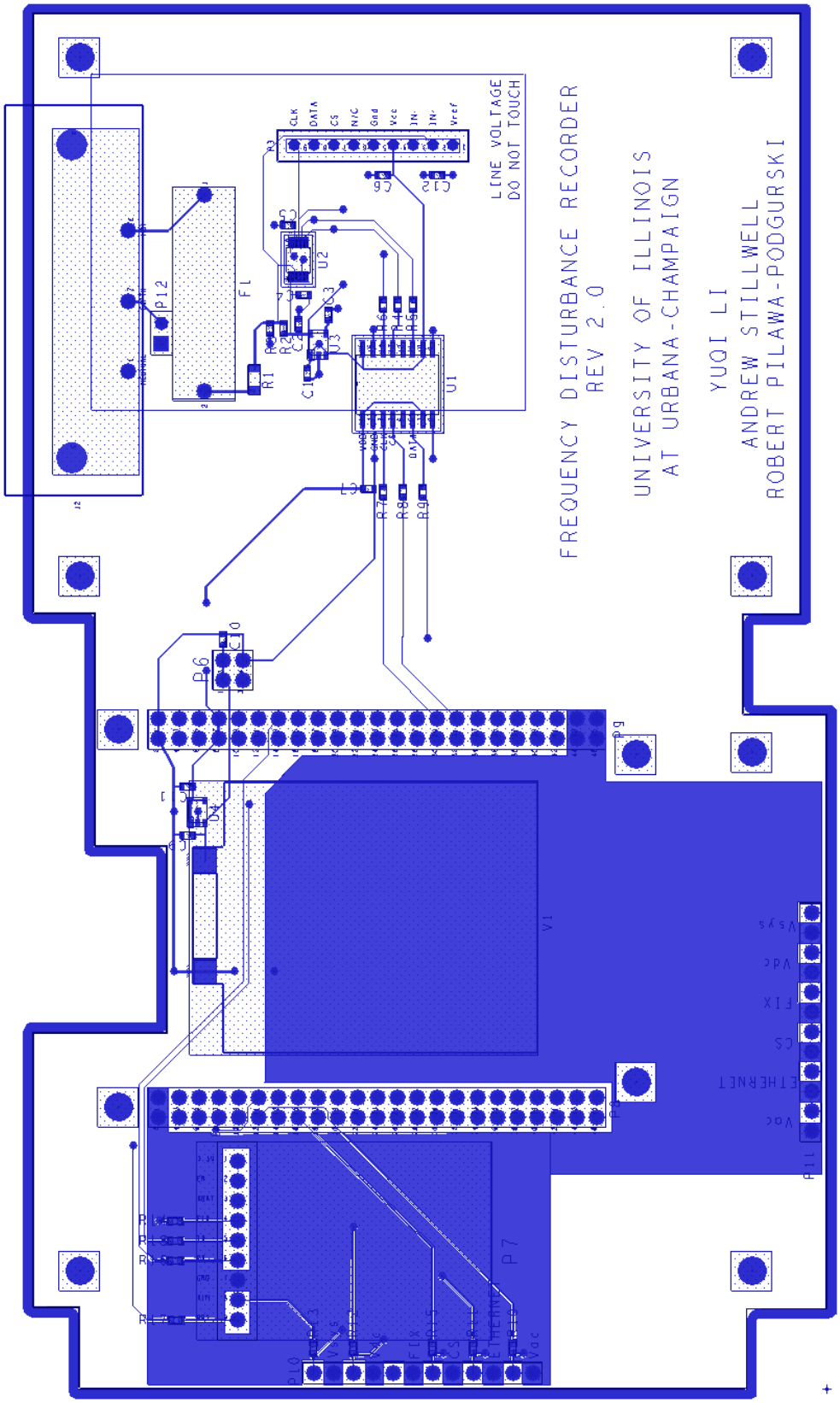

Figure B.1: PCB layout top view.

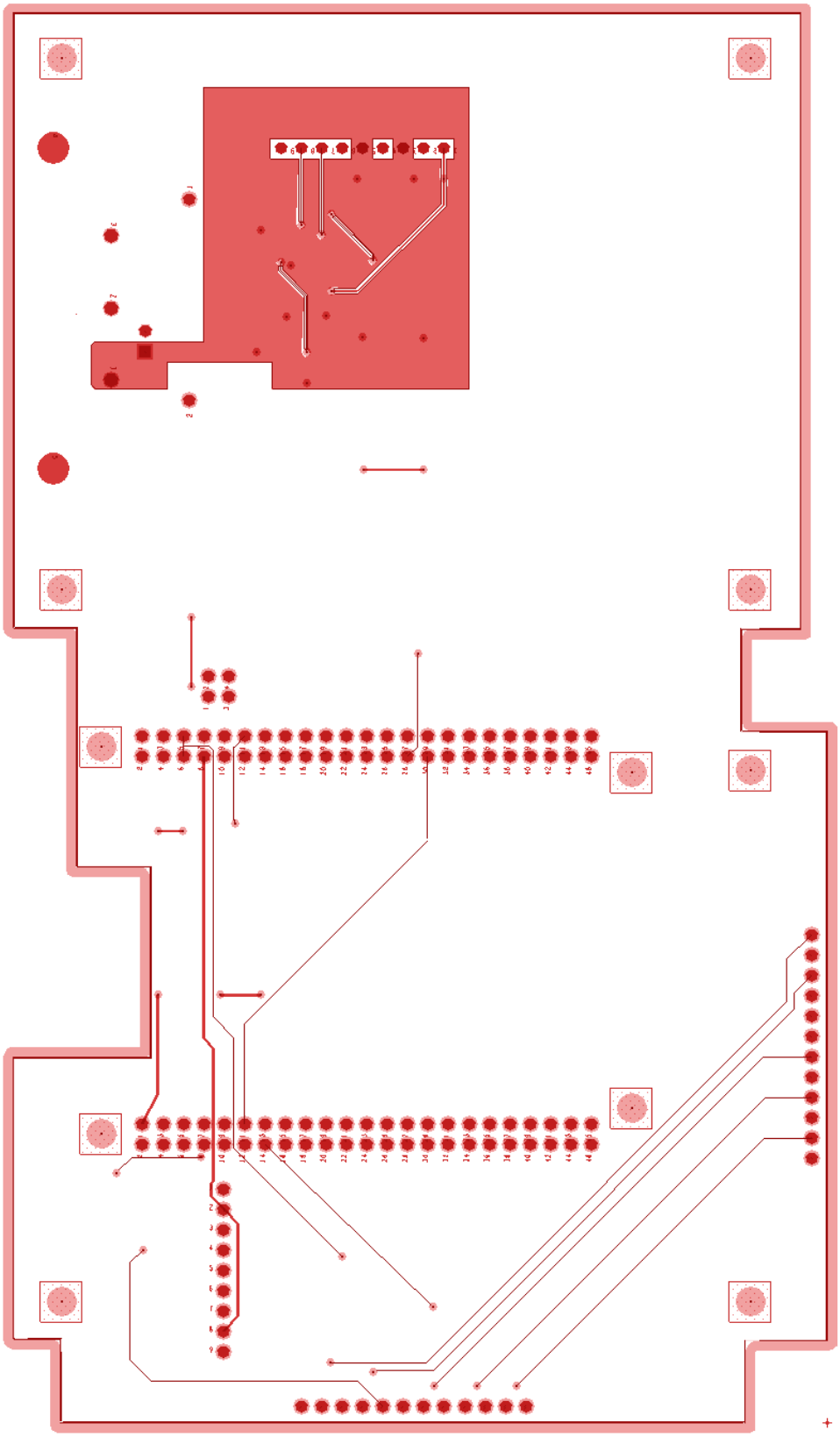

Figure B.2: PCB layout bottom view.

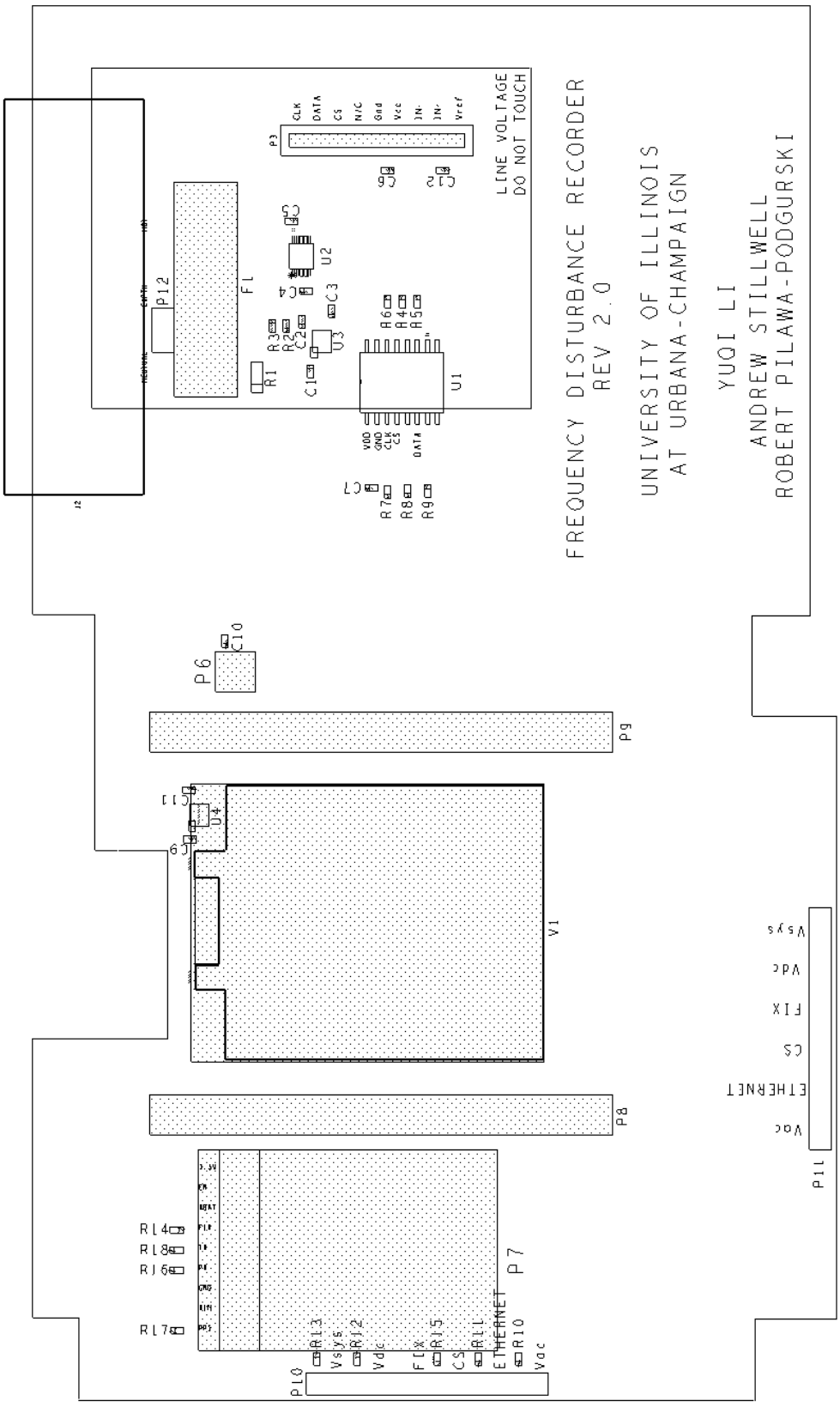

Figure B.3: Top silkscreen.

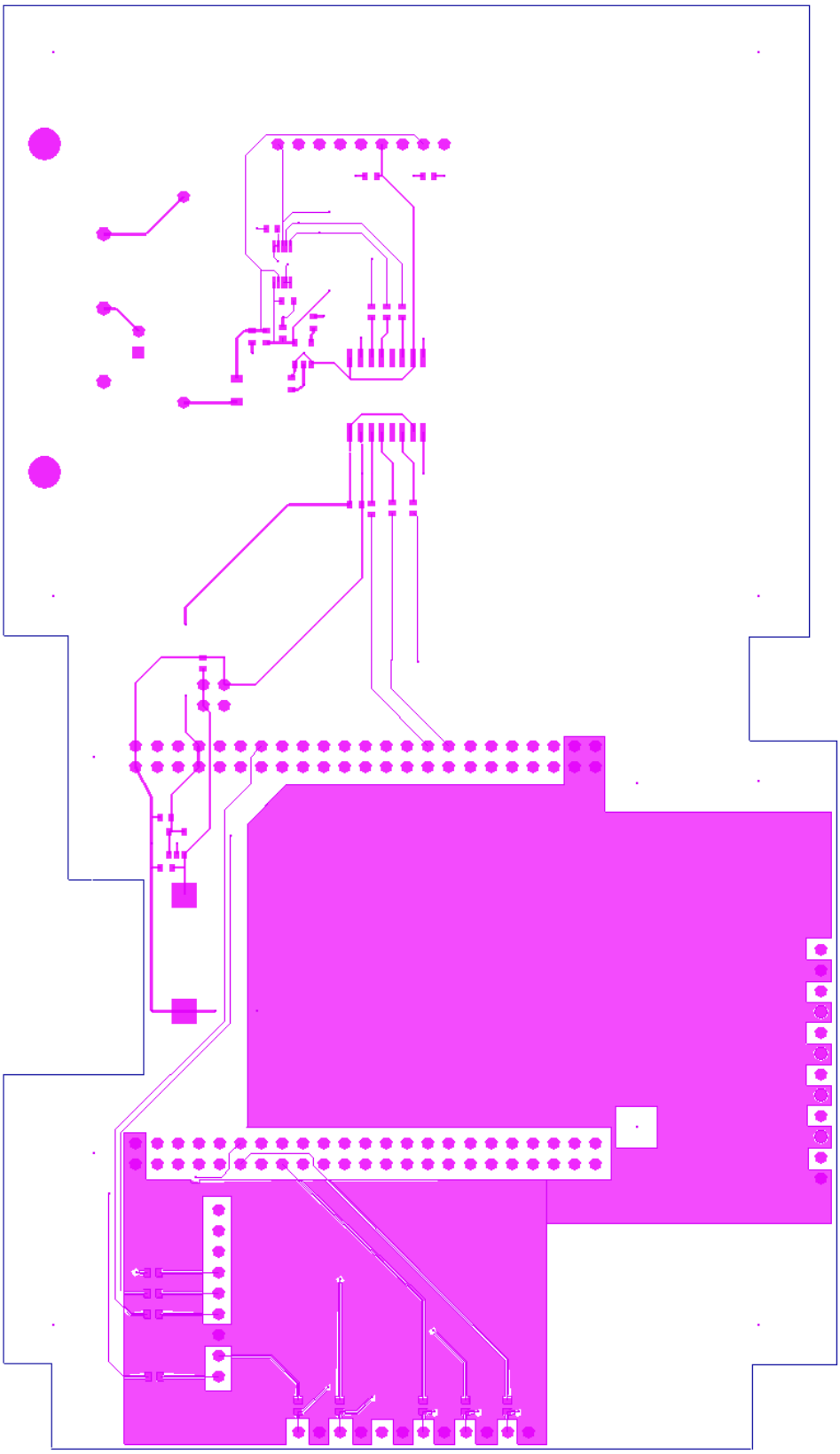

Figure B.4: Top-layer copper.

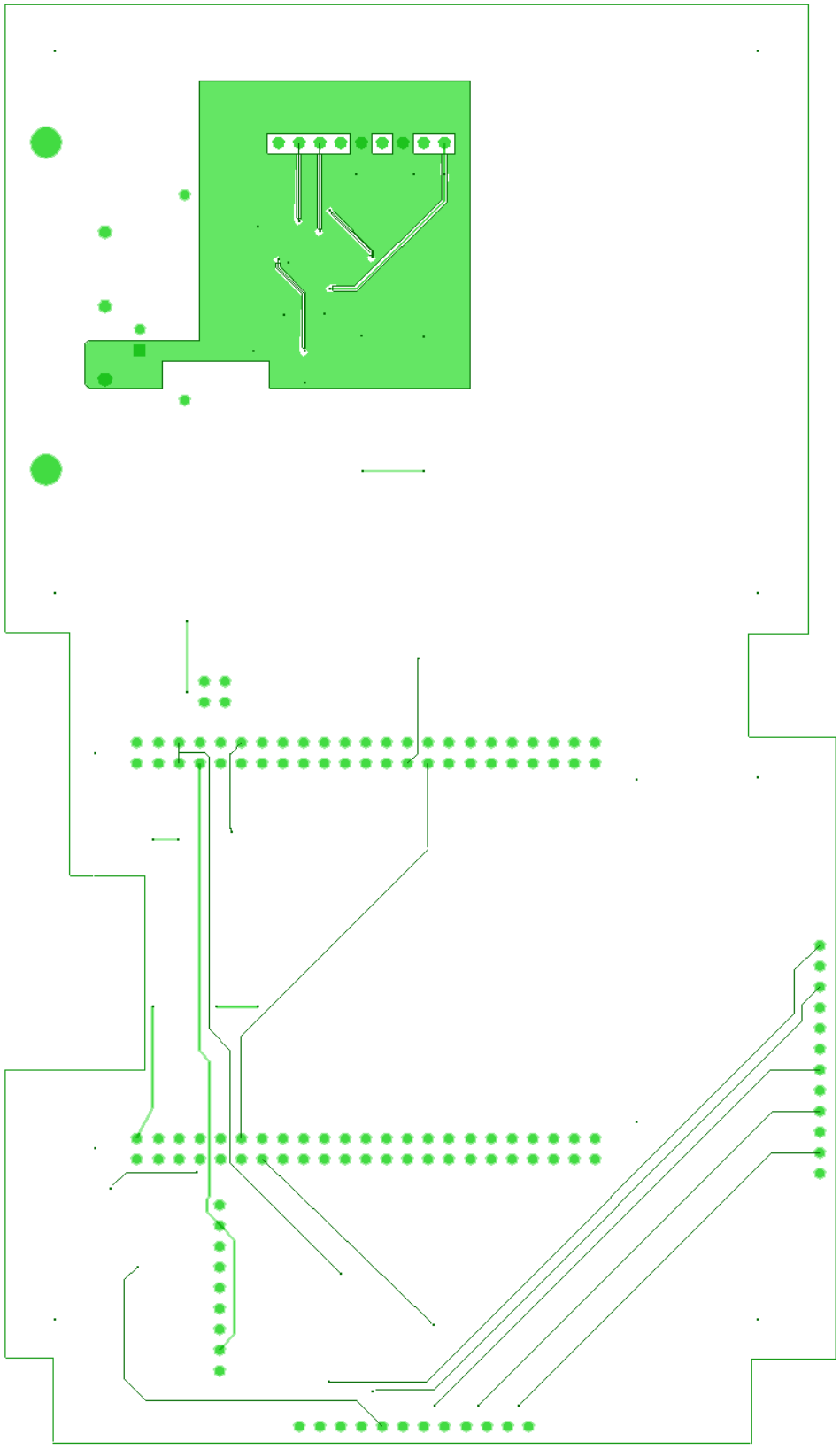

Figure B.5: Bottom-layer copper.

APPENDIX C

PROTOTYPE PHOTOS FOR DGAP REV 2.0

The device photos from multiple views are shown in Figs. C.1 – C.9

Figure C.1: Top view of the fully assembled DGAP.

Figure C.2: Front view of the fully assembled DGAP.

Figure C.3: Side view of the fully assembled DGAP.

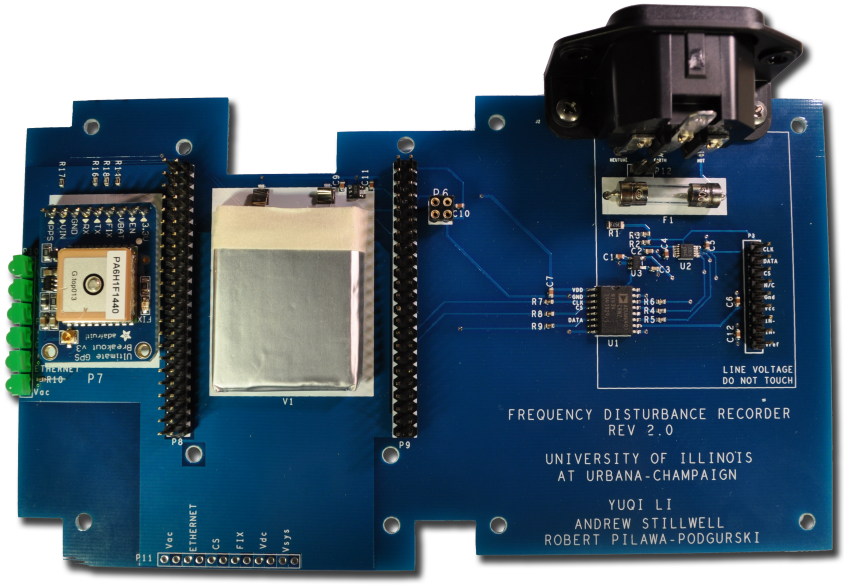

Figure C.4: Top view of the board without BBB.

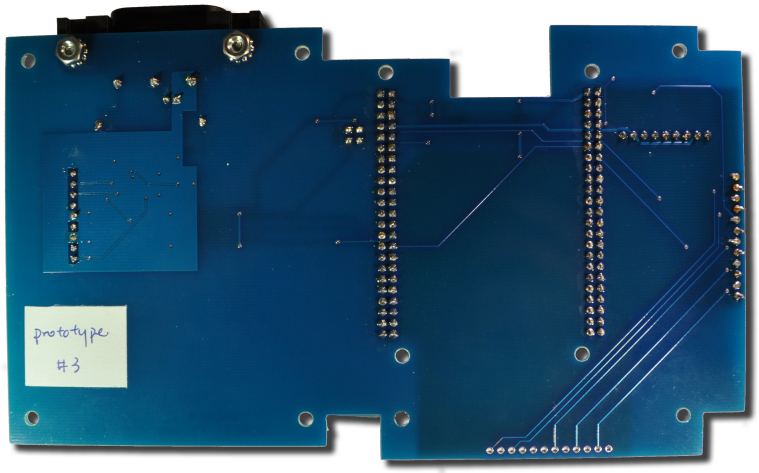

Figure C.5: Bottom view of the board without BBB.

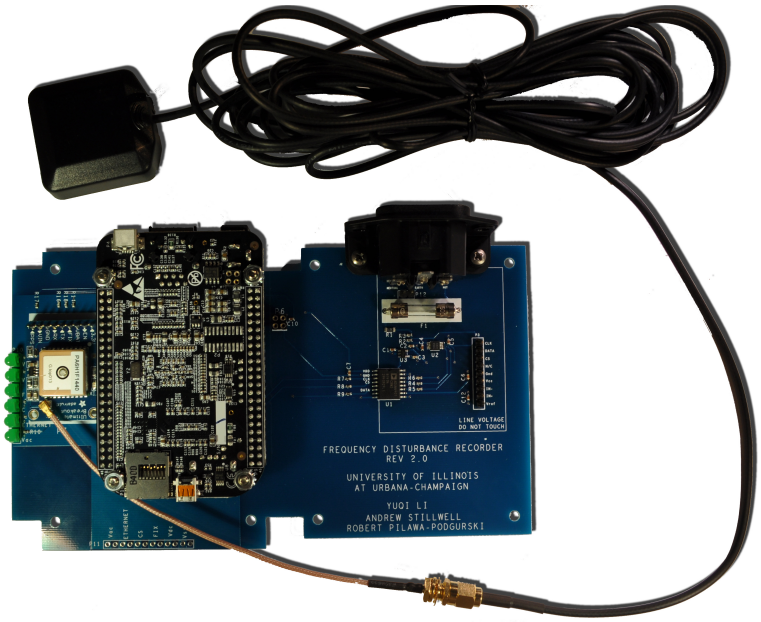

Figure C.6: Top view of the board with BBB and external antenna attached.

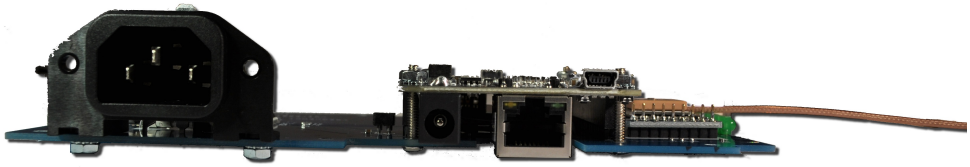

Figure C.7: Front view of the board with BBB and external antenna attached.

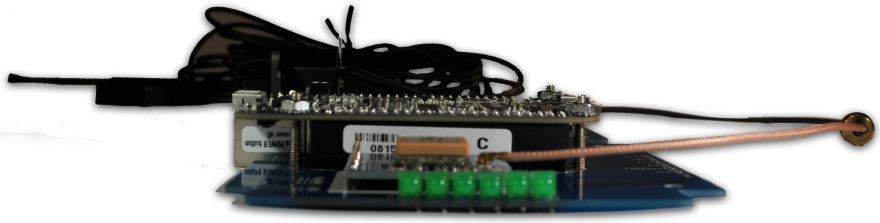

Figure C.8: Side view of the board with BBB and external antenna attached.

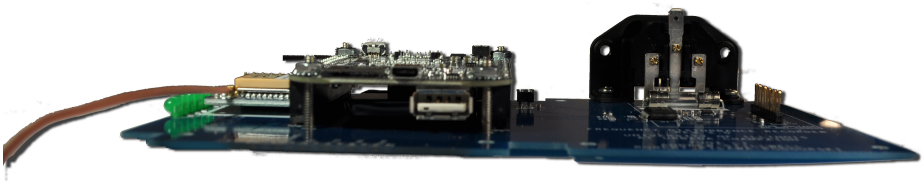

Figure C.9: Rear view of the board with BBB and external antenna attached.

APPENDIX D

COMPONENT LISTING FOR DGAP REV 2.0

The components mentioned in Section 2.2 are listed in Table D.1.

Table D.1: Component List for the Distributed Grid Analytics Platform

Component	Model	Value	Lg Qty Price ($)
Single-board Computer	Beaglebone Black		55.95
GPS	Adafruit Ultimate GPS Breakout		38.99
Antenna Adapter	Adafruit SMA to uFL RF Adapter Cable		3.95
Antenna	Adafruit GPS Antenna SMA		12.95
Linear Regulator	LP2985-33DBVR		0.22
Battery Charging Mgmt Chip	MCP73811		0.39
Digital & Power Isolator	ADuM6401		7.60
ADC	ADC141S626		3.09
Power Outlet Connector	Inlet C14		1.83
Li-ion Battery	PL-403144		3.56
Fuse	F3469		0.59
LED	78-TLPG5600		0.11
C	0603	1 μF	
	0603	2.2 μF	
	0603	10 nF	
R	1206	357 kΩ	
	0603	6.19 kΩ	
	0603	5 Ω	
Container	HM1056		7.28
		Total Cost Per Device:	136.51

APPENDIX E

EXPERIMENTAL DATA FOR DGAP REV 2.0

In Section 4.2, to test the reliability of the DGAP for input higher or lower than the nominated 120 V_{rms} 60 Hz waveform, frequencies in the range of 59.0 Hz - 61.0 Hz (actual frequency provided by power supply) are measured and the outputs (frequency and voltage read) are listed in Table E.1 and Table E.2 respectively.

Table E.1: Frequency Measurements and Reading Error

Actual Frequency (Hz)	Frequency Read (Hz)	Error (Hz)	Error (%)
62.0	62.1556	-0.1556	0.250968
61.0	61.0800	-0.0800	0.131148
60.9	60.9718	-0.0718	0.117898
60.8	60.8638	-0.0638	0.104934
60.7	60.7562	-0.0562	0.092586
60.6	60.6438	-0.0438	0.072277
60.5	60.5393	-0.0393	0.064959
60.4	60.4318	-0.0318	0.052649
60.3	60.3234	-0.0234	0.038806
60.2	60.2155	-0.0155	0.025748
60.1	60.1074	-0.0074	0.012313
60.0	60.0000	0.0000	0.000000
59.9	59.8924	0.0076	0.012688
59.8	59.7838	0.0162	0.027090
59.7	59.6758	0.0242	0.040536
59.6	59.5678	0.0322	0.054027
59.5	59.4597	0.0403	0.067731
59.4	59.3511	0.0489	0.082323
59.3	59.2431	0.0569	0.095953
59.2	59.1351	0.0649	0.109628
59.1	59.0217	0.0783	0.132487
59.0	58.9195	0.0805	0.136441
58.0	57.8425	0.1575	0.271552

Table E.2: V_{rms} Measurements and Reading Error

Actual V_{rms} (V)	Measured V_{rms} (V)	Error (V)	Error (%)
110.3	110.3	0	0.000000
111.3	111.3	0	0.000000
112.3	112.3	0	0.000000
113.3	113.3	0	0.000000
114.3	114.3	0	0.000000
115.3	115.3	0	0.000000
116.3	116.3	0	0.000000
117.3	117.2	0.1	0.085251
118.3	118.2	0.1	0.084531
119.3	119.2	0.1	0.083822
120.3	120.2	0.1	0.083126
121.3	121.2	0.1	0.082440
122.3	122.1	0.2	0.163532
123.3	123.1	0.2	0.162206
124.3	124.1	0.2	0.160901
125.3	125.1	0.2	0.159617
126.3	126.0	0.3	0.237530
127.3	127.0	0.3	0.235664
128.3	128.0	0.3	0.233827
129.3	128.9	0.4	0.309358
130.3	129.9	0.4	0.306984

APPENDIX F

DGAP CODE IMPLEMENTED IN BEAGLEBONE BLACK

In this appendix, the code implemented in Beaglebone Black at PRU and ARM is presented. Contributed mainly by Andrew Stillwell.

F.1 Assembly Code for Programmable Real-time Unit

In this section, the assembly code for the PRU, which samples data from ADC and transfers the data to the shared memory, is included.

F.1.1 Data Acquiring

The following assembly code on PRU 0 acquires data from the ADC, and transfers the data to shared memory.

```
1  .setcallreg r29.w0
2
3  .origin 0
4  .entrypoint MAIN
5
6  #include "pru.hp"
7  #include "pructl.h"
8
9
10
11 #define RECORDS 20000
12 #define GPIO1 0x4804c000
13 #define GPIO_CLEARDATAOUT 0x190
14 #define GPIO_SETDATAOUT 0x194
```

```
15
16 // sample freq is 2*T1+12 (high time) + 64*T2 + 178 (low / CS
       enable time)
17 // this is dependent on all the calls made... so try not to add/
       remove unless you want to recount
18 #define Time1 8625
19 #define Time2 40
20 // T1 = 3625, T2 = 40 gives 20kS/s sample rate and 2.5MHz SClk
21 //T1 = 8625, T2 = 40 -> 10kS/s
22 //T1 = 18625, T2 = 40 -> 5kS/s
23
24
25
26 // *** LED routines, so that LED USR0 can be used for some
       simple debugging
27 // *** Affects: r2, r3
28 .macro LED_OFF
29     MOV r2, 1<<21
30     MOV r3, GPIO1 | GPIO_CLEARDATAOUT
31     SBBO r2, r3, 0, 4
32 .endm
33
34 .macro LED_ON
35     MOV r2, 1<<21
36     MOV r3, GPIO1 | GPIO_SETDATAOUT
37     SBBO r2, r3, 0, 4
38 .endm
39
40
41
42
43
44
45 .macro UDEL
46 UDEL:
47     MOV r4, Time2
48 UDEL1:
49     SUB r4, r4, 1
50     QBNE UDEL1, r4, 0 // loop if we've not finished
51 .endm
52
53 .macro MDEL
54 MDEL:
```

```
55      MOV r4, Time1
56  MDEL1:
57      SUB r4, r4, 1
58      QBNE MDEL1, r4, 0 // loop if we've not finished
59  .endm
60
61  .macro DEL
62  DEL:
63      MOV r4, 10000
64  DEL1:
65      SUB r4, r4, 1
66      QBNE DEL1, r4, 0 // loop if we've not finished
67  .endm
68
69
70
71
72
73
74
75
76  //-----------------------------------------------
77  MAIN:
78
79      // C24 - Local memory
80      // Configure the block index register for PRU0 by setting
81      // c24_blk_index[7:0] and c25_blk_index[7:0] field to 0x00
        and 0x00,
82      // respectively.  This will make C24 point to 0x00000000 (
        PRU0 DRAM) and
83      // C25 point to 0x00002000 (PRU1 DRAM).
84      MOV       r0, 0x00000000
85      MOV       r1, CTBIR_0
86      ST32      r0, r1
87
88  //
89  //    SRAM
90  //              +-- rTailPtr        +--- rHeadPtrPtr
91  //              |                   |   + -- rDrmOffsetPtrPtr
92  //              |                   |   |
93  //              |                   |   |
94  //              V                   V   V
95  //    +-------------------------+-------------------------+
```

```
96  //     |                              |                              |
97  //     +------------------------------+------------------------------+
98  //      <---- rTailMask ------------->
99  //
100 //     DRAM
101 //
102 //     +-- rDrmBasePtr
103 //     |
104 //     |<----- rDrmOffset -----|
105 //     |                       |
106 //     V                       V
107 //
        +-------------------------------------------------------------------+

108 //     |
        |
109 //
        +-------------------------------------------------------------------+

110 //      <------------------- rDrmOffsetMask
        ------------------------------>
111 //
112
113 #define MAGIC_NUMBER          0xbabe7176
114
115
116 #define     rHeadPtrPtr   r10
117     MOV         rHeadPtrPtr, (0x0000 + PRU0_OFFSET_SRAM_HEAD)   //
        0x1000
118
119 #define     rDrmOffsetMask r11
120     MOV         rDrmOffsetMask, 0x0003ffff
121
122 #define     rDrmOffset      r12
123     MOV         rDrmOffset, 0x0000
124
125 #define     rDrmBasePtr     r13
126     MOV         rDrmBasePtr, (0x0000 + PRU0_OFFSET_DRAM_PBASE)
127     LD32        rDrmBasePtr, rDrmBasePtr
128
129 #define     rDrmOffsetPtrPtr r14
130     MOV         rDrmOffsetPtrPtr, (0x0000 + PRU0_OFFSET_DRAM_HEAD)
131
```

```
132
133   LED_ON
134
135 loop_label:
136     // put pps code here, sample for 1 second
137     MOV r7, RECORDS // load records into r7
138 READLOOP:
139     MDEL
140     MOV r0, 16  //spi bits counter
141     MOV r0, 16  // do this again for timing purposes
142     CLR r30.t0  // enable CS
143     MOV r3, 0 // sample storage
144
145 SPIREAD:
146     UDEL
147     CLR r30.t1  // SCLK Falling edge
148     UDEL
149     SET r30.t1 // SCLK Rising edge
150     MOV r2, r31  // read right after Rising edge
151     AND r2, r2, 8 // mask all but the 4th bit - which should be
    the MISO
152     LSR r2, r2, 3   // shift the MISO bit to the LSB
153     SUB r0, r0, 1   //decrement SPI bit counter
154     LSL r2, r2, r0  // shift MISO bit to position based on
    number of spi bits read
155     OR r3, r3, r2   // OR with previous values
156     QBNE SPIREAD, r0, 0   // loop until all 16 bits have been
    read
157     SET r30.t0        // disable CS
158     // should have a 16 bit sample in the lower 16 bits of r3 at
     this point
159
160     // Store results at ddr dst addr
161     SBBO      r3, rDrmBasePtr, rDrmOffset, 4         // 1+8 = 9
    + latency
162     ADD       rDrmOffset, rDrmOffset, 4                  // 1
163     AND       rDrmOffset, rDrmOffset, rDrmOffsetMask      // 1
164
165     // Store DDR fifo offset into sram
166     ST32      rDrmOffset, rDrmOffsetPtrPtr                // 2
167
168
169     SUB r7, r7, 1
```

```
170     QBNE READLOOP, r7, 0 // loop if we've not finished
171
172     // Do it all over again
173     JMP        loop_label                                    // 1
```

code/prucode_ring.p

F.1.2 Time Synchronization

The following assembly code on PRU 1 counts the number of micro-controller timebase ticks between PPS GPS signals.

```
1  //
2  //
3  // This source code is available under the "Simplified BSD
       license".
4  //
5  // Copyright (c) 2013, J. Kleiner
6  // All rights reserved.
7  //
8  // Redistribution and use in source and binary forms, with or
       without
9  // modification, are permitted provided that the following
       conditions are met:
10 //
11 // 1. Redistributions of source code must retain the above
       copyright notice,
12 //    this list of conditions and the following disclaimer.
13 //
14 // 2. Redistributions in binary form must reproduce the above
       copyright
15 //    notice, this list of conditions and the following
       disclaimer in the
16 //    documentation and/or other materials provided with the
       distribution.
17 //
18 // 3. Neither the name of the original author nor the names of
       its contributors
19 //    may be used to endorse or promote products derived from
       this software
20 //    without specific prior written permission.
```

```
21 //
22 // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
       CONTRIBUTORS
23 // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT
       NOT
24 // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
       FITNESS FOR
25 // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
       COPYRIGHT
26 // HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
       INCIDENTAL,
27 // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
       NOT LIMITED
28 // TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
       DATA,
29 // OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
       ANY THEORY
30 // OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
31 // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
       THE USE
32 // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
       DAMAGE.
33 //
34 //
35
36 .setcallreg r29.w0
37
38 .origin 0
39 .entrypoint MAIN1
40
41 #include "pru.hp"
42 #include "pructl.h"
43
44 .macro DEL
45 DEL:
46     MOV r4, 50000000
47 DEL1:
48     SUB r4, r4, 1
49     QBNE DEL1, r4, 0 // loop if we've not finished
50 .endm
51
52
53
```

```
54 //-----------------------------------------------------
55 MAIN1:
56     // Enable OCP master port
57     LBCO      r0, CONST_PRUCFG, 4, 4
58     CLR       r0, r0, 4  // Clear SYSCFG[STANDBY_INIT] to enable
       OCP master port
59     SBCO      r0, CONST_PRUCFG, 4, 4
60
61 //
62 //   SRAM
63 //            +-- rHeadPtr        +--- rHeadPtrPtr
64 //            |                   |   + -- rSpinPtr
65 //            |                   |   |
66 //            |                   |   |
67 //            V                   V   V
68 //    +-------------------------+-------------------------+
69 //    |                         |                         |
70 //    +-------------------------+-------------------------+
71 //     <---- rHeadMask --------->
72 //
73
74 #define rHeadPtr     r3
75     MOV       rHeadPtr,     (0x2000)
76
77 #define rHeadMask    r4
78     MOV       rHeadMask,    0x2fff
79
80 #define rHeadPtrPtr  r5
81     MOV       rHeadPtrPtr,  (0x2000 + PRU0_OFFSET_SRAM_HEAD)  //
       0x3000
82
83 #define rSpinPtr     r6
84     MOV       rSpinPtr,     (0x2000 + PRU0_OFFSET_SPIN_COUNT)
      // 0x3008
85
86 #define rSpinCnt     r7
87     LD32      rSpinCnt, rSpinPtr
88
89 #define rTBCnt     r8   // register to count timebase ticks
90     MOV rTBCnt, 100000003
91
92 #define rPPSCnt     r9    // register to count PPS received
93     MOV rPPSCnt, 0
```

```
94
95
96 #define rSampMask     r12
97     MOV         rSampMask, 0xffff
98 #define MAGIC_NUMBER         0xbabe7176
99
100   MOV      r14, MAGIC_NUMBER
101      // Update sram with new head
102      ST32       r14, rHeadPtrPtr     // dump magic number in the
      SRAM so we know nothing is there
103
104 init_label:
105   QBNE init_label, r31, 0x3C // loop if we've not finished
106   ADD rPPSCnt, rPPSCnt, 1
107   ST32 rPPSCnt, rSpinPtr
108   DEL              // wait .5 seconds for the PPS to go low
109 wait_label:
110   ADD rTBCnt, rTBCnt, 2
111   QBNE wait_label, r31, 0x3C // loop if we've not finished
112   ST32 rTBCnt, rHeadPtrPtr
113   ADD rPPSCnt, rPPSCnt, 1
114   ST32 rPPSCnt, rSpinPtr
115   DEL              // wait .5 seconds for the PPS to go low
116   MOV rTBCnt, 100000006
117   JMP wait_label
118
119
120
121
122
123
124      // Goto top of main loop
125      JMP        wait_label                        // 1
```

code/prucode_pps.p

F.2 C Code for PRU Initialization and Data Transfer Between Shared Memory and Python Code

The following C code initializes the PRUs, loads the code, reads the ADC data from shared memory, and transfers it to the Python code described in Appendix. F.3.

```c
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>

#include <prussdrv.h>
#include <pruss_intc_mapping.h>
#include "pructl.h"
#include "unixSocketClient.h"

//-----------------------------------------------------
static void *ddrMem;
static void *pru0DataMem;
static void *pru1DataMem;

//-----------------------------------------------------
void show_hex( unsigned char *bf, int bytes )
{
    int idx;
    unsigned short w;
    volatile unsigned char *vbf;

    vbf = bf;

    idx = 0;
    while( idx < bytes ){
        if( 0==(idx%16) ) printf("\n0x%08x ",idx);
        printf("%02x%02x ",vbf[idx+1],vbf[idx]);
        idx+=2;
    }
    printf("\n");
}

```

```
36 //---------------------------------------------------
37 void show_csv( unsigned char *bf, int bytes )
38 {
39     int idx;
40     unsigned short w;
41     volatile unsigned char *vbf;
42 
43     idx = 0;
44     while( idx < bytes ){
45         w = ( (unsigned short)(vbf[idx+1])<<8 ) | (unsigned
    short)(vbf[idx]);
46         // w = w >> 2;
47         printf("%d, %d\n",idx,w);
48         idx+=2;
49     }
50     printf("\n");
51 }
52 
53 //---------------------------------------------------
54 unsigned int getu32( void *ptr, int offset )
55 {
56     unsigned int v;
57 
58     v = *( (unsigned int*)( ((volatile unsigned char*)ptr)+
    offset ));
59 
60     return( v );
61 }
62 
63 //---------------------------------------------------
64 void setu32( void *ptr, int offset, unsigned int v )
65 {
66     *( (unsigned int*)( ((volatile unsigned char*)ptr)+offset ))
     = v;
67 }
68 
69 //---------------------------------------------------
70 int main(int argc, char *argv[]  )
71 {
72     unsigned int ret;
73     tpruss_intc_initdata pruss_intc_initdata =
    PRUSS_INTC_INITDATA;
74     void *vptr;
```

```
75      void *pptr;
76    unsigned int dRead    = 0;
77    unsigned int dWrite   = 0;
78    unsigned int dDDR   = 0;
79    unsigned int ddrMask    = 0x0003ffff;   // this is based on
       the default DDR size, update if that is changed
80    FILE* outfile;
81
82    // Necessary variables for socket operation
83    char socketPath[] = "/tmp/testSocket";
84    int s;
85
86    // TODO: a lot of these are not used, left over from sample
       code
87      int idx;
88      int bLoad0    = 0;
89      int bLoad1    = 0;
90      int bShow     = 0;
91      int bStop     = 0;
92      int bSnap     = 0;
93      int bClean    = 0;
94      int bCsv      = 0;
95      int bShowSram = 0;
96      int bSetSpin  = 0;
97    int bRun1     = 0;
98      int offset    = 0;
99      int spinCount = 100;
100   int i = 0;
101   int k = 0;
102   int fs = 10000;          //ADC sampling frequency - may change
       this to an input
103   int fPMU = 20;           // PMU sample rate - up to 20S/s for
       60hz
104   int sProc = fs/fPMU;
105   int sBuff[sProc];
106   char cmd[100];
107   char temp[32];
108   char *data = NULL;
109   size_t len = 0;
110   ssize_t dataLen;
111   int error = 0;
112
113
```

```
114
115
116
117
118   outfile=fopen("data.csv", "w");
119
120   printf("**********");
121   // create a socket which is identified by an integer value s
122   createSocket(&s);
123   // connect this socket to the user defined socket path (akin
       to an ip address and port),
124   // on which the python server is listening
125   connectToSocket(&s, socketPath);
126
127     idx=1;
128     while( idx< argc ){
129         if( 0==strcmp(argv[idx],"-load0") ){
130             bLoad0 = 1;
131         }
132
133         if( 0==strcmp(argv[idx],"-load1") ){
134             bLoad1 = 1;
135         }
136
137         if( 0==strcmp(argv[idx],"-show") ){
138             bShow = 1;
139         }
140         if( 0==strcmp(argv[idx],"-stop") ){
141             bStop = 1;
142         }
143         if( 0==strcmp(argv[idx],"-snap") ){
144             bSnap = 1;
145         }
146         if( 0==strcmp(argv[idx],"-clean") ){
147             bClean = 1;
148         }
149         if( 0==strcmp(argv[idx],"-csv") ){
150             bCsv = 1;
151         }
152     if( 0==strcmp(argv[idx],"-run1") ){
153             bRun1 = 1;
154         }
155         if( 0==strcmp(argv[idx],"-sram") ){
```

```
156             char ch;
157             bShowSram = 1;
158             offset    = strtol( argv[idx+1], &ch, 0 );
159         }
160         if( 0==strcmp(argv[idx],"-spin") ){
161             char ch;
162             bSetSpin = 1;
163             spinCount= strtol( argv[idx+1], &ch, 0 );
164         }
165         idx++;
166     }
167     printf("load0 = %d\n",bLoad0);
168     printf("load1 = %d\n",bLoad1);
169     printf("show  = %d\n",bShow);
170     printf("stop  = %d\n",bStop);
171     printf("show sram  = %d,%d\n",bShowSram,offset);
172     printf("spin  = %d,%d\n",bSetSpin,spinCount);
173 
174   // code that programatically loads the device tree overlay and
       starts the PRUs
175   //only need to run this once
176   // TODO: read the slots file and find out if the overlay has
       already been loaded
177 
178   sprintf(cmd, "echo BB-BONE-HSADC > /sys/devices/bone_capemgr
       .9/slots");
179   system(cmd);
180   sleep(1);
181 
182 
183 
184 
185 
186     /* Initialize the PRU */
187     printf("\tINFO: pruss init\n");
188     prussdrv_init ();
189 
190     /* Open PRU Interrupt */
191     printf("\tINFO: pruss open\n");
192     ret = prussdrv_open(PRU_EVTOUT_0);
193     if (ret)
194     {
195         printf("prussdrv_open open failed\n");
```

```
196         return (ret);
197     }
198 
199     /* Get the interrupt initialized */
200     printf("\tINFO: pruss intc init\n");
201     prussdrv_pruintc_init(&pruss_intc_initdata);
202 
203     /* Initialize example */
204     printf("\tINFO: mapping memory \n");
205     prussdrv_map_prumem (PRUSS0_PRU0_DATARAM, &pru0DataMem);
206     prussdrv_map_prumem (PRUSS0_PRU1_DATARAM, &pru1DataMem);
207 
208     /* Setup DRAM memory */
209     printf("\tINFO: setup mem \n");
210     // line 215 extram_phys_base is phys and read from sysfs
211     // line 231 extram_base      is virt and mmapped
212 
213     prussdrv_map_extmem( &vptr );
214     printf("V extram_base = 0x%08x\n",(unsigned int) vptr);
215     ddrMem = vptr;
216 
217     pptr = (void*)prussdrv_get_phys_addr( vptr );
218     printf("P extram_base = 0x%08x\n",(unsigned int) pptr);
219 
220     if( bClean ){
221   memset( ddrMem, 0xa5, 0x100 );
222     }
223 
224 
225 
226         // Set the number of bytes to capture
227         setu32(pru0DataMem,PRU0_OFFSET_SRAM_HEAD,0);
228 
229         // Give the pru dram/ddr memory base
230         setu32(pru0DataMem,PRU0_OFFSET_DRAM_PBASE,
231                             prussdrv_get_phys_addr(ddrMem) );
232 
233   // Set spin count based on specified or defaul
234         setu32(pru0DataMem, PRU0_OFFSET_SPIN_COUNT, spinCount );
235 
236         // Place marker s
237         setu32(pru0DataMem, PRU0_OFFSET_RES1,       0xdeadbeef );
238         setu32(pru0DataMem, PRU0_OFFSET_SRAM_TAIL, 0x0 );
```

```
239         setu32(pru0DataMem, PRU0_OFFSET_DRAM_HEAD, 0x0 );
240         setu32(pru0DataMem, PRU0_OFFSET_RES2,      0xbabedead );
241         setu32(pru0DataMem, PRU0_OFFSET_RES3,      0xbeefcafe );
242
243         show_hex( ((unsigned char*)pru0DataMem)+0x1000, 32 );
244
245         printf("\tINFO: loading pru 0 code \n");
246         prussdrv_exec_program (0, "prucode_ring.bin");
247
248
249   // NOTE: for some reason we have to load PRU 1 first,
       otherwise PRU 0 won't initialize.
250   // After loading PRU 1 once, do not have to load it again
251
252         printf("\tINFO: loading pru 1 code \n");
253         prussdrv_exec_program (1, "prucode_pps.bin");
254
255
256
257   sleep(2);
258
259         sprintf(temp, "%lu", getu32(pru0DataMem,
       PRU0_OFFSET_SRAM_HEAD));
260         printf(" ticks cnt %s\n", temp);
261         sendData(&s, &temp, 10);
262         sprintf(temp, "%08x", getu32(pru0DataMem,
       PRU0_OFFSET_SPIN_COUNT));
263         printf(" pps cnt %s\n", temp);
264         sendData(&s, &temp, 10);
265
266
267
268
269   i=0;
270   while(error == 0){       // TODO: find a better condition for
       this and how to reset gracefully
271     dWrite = 0;
272     while(dWrite < sProc){
273       dDDR = getu32(pru0DataMem,PRU0_OFFSET_DRAM_HEAD);
274       //printf("PRU->dram    : 0x%08x\n",dDDR );
275
276       while(dRead!= dDDR & dWrite < sProc ){
277       //   sBuff[dWrite+1] = getu32(ddrMem,dRead);
```

```
278           sprintf(temp, "%ld", getu32(ddrMem,dRead));
279         //  sprintf(temp, "%ld", i);     //debug
280           data = &temp;
281         //  fprintf(outfile, "%d\n", sBuff[dWrite+1]);
282           sendData(&s, &temp, 4);
283
284           dRead = (dRead+4) & ddrMask;
285           i++;
286           dWrite++;
287         }
288           // write out PPS count and ticks once per second
289           if(i%fs == 0){
290             sprintf(temp, "%lu", getu32(pru0DataMem,
    PRU0_OFFSET_SRAM_HEAD));
291         //  printf(" ticks cnt %s\n", temp);
292   //DB         sendData(&s, &temp, 10);
293             sprintf(temp, "%lu", getu32(pru0DataMem,
    PRU0_OFFSET_SPIN_COUNT));
294         //  printf(" pps cnt %s\n", temp);
295   //DB         sendData(&s, &temp, 10);
296           //printf(" samples written %d\n", dWrite);
297           i = 0;
298   // TODO: add error flag for things like buffer over run, no
    PPS lock.... other errors
299         }
300
301     }
302   /*  // error checking code for an incremental count pattern
303     for(i = 1;i<sProc;i++){
304       if( sBuff[i+1] - sBuff[i] != 1){
305         printf("error found\n");
306       }
307     }
308   */
309
310   }
311
312
313
314
315
316
317
```

```
318
319
320     if( bCsv ){
321         show_csv( (unsigned char*)ddrMem, 8192 );
322     }
323
324     if( bSnap ){
325         show_hex( ((unsigned char*)pru0DataMem)+0x1000, 32 );
326         show_hex( ((unsigned char*)pru0DataMem)+0x0000, 0x100 );
327         show_hex( (unsigned char*)ddrMem,               0x100 );
328     }
329
330
331
332     if( bShow ){
333   unsigned int v;
334
335   printf("\n");
336   printf("PRU  head    : 0x%08x\n",
337                                 getu32(pru0DataMem,
     PRU0_OFFSET_SRAM_HEAD) );
338   printf("DRAM paddr   : 0x%08x\n",
339                                 getu32(pru0DataMem,
     PRU0_OFFSET_DRAM_PBASE) );
340   printf("PRU spin     : 0x%08x\n",
341                                 getu32(pru0DataMem,
     PRU0_OFFSET_SPIN_COUNT) );
342   printf("Res1         : 0x%08x\n",
343                                 getu32(pru0DataMem,
     PRU0_OFFSET_RES1) );
344   printf("PRU tail     : 0x%08x\n",
345                                 getu32(pru0DataMem,
     PRU0_OFFSET_SRAM_TAIL) );
346   printf("PRU->dram    : 0x%08x\n",
347                                 getu32(pru0DataMem,
     PRU0_OFFSET_DRAM_HEAD) );
348   printf("Res2         : 0x%08x\n",
349                                 getu32(pru0DataMem,
     PRU0_OFFSET_RES2) );
350   printf("Res3         : 0x%08x\n",
351                                 getu32(pru0DataMem,
     PRU0_OFFSET_RES3) );
352
```

```
353     printf("SRAM excerpt");
354           show_hex( ((unsigned char*)pru0DataMem)+0x0000, 0x20 );
355
356     printf("DRAM excerpt");
357           show_hex( (unsigned char*)ddrMem,               0x20 );
358       }
359
360       if( bShowSram ){
361           printf("Relative to 0x%08x",offset);
362           show_hex( ((unsigned char*)pru0DataMem)+offset, 0x100 );
363       }
364
365       if( bSetSpin ){
366           setu32(pru0DataMem, PRU0_OFFSET_SPIN_COUNT, spinCount );
367       }
368
369
370           /* Disable PRU and close memory mapping*/
371           printf("\tINFO: closing pru \n");
372           prussdrv_pru_disable (0);
373           prussdrv_pru_disable (1);
374           prussdrv_exit ();
375
376
377
378     close(s);
379     printf("close\n");
380     printf("free\n");
381
382     fclose(outfile);
383       return(0);
384
385 }
```

code/ringtest.c

F.3 Python Code in ARM CPU

This section includes the main Python application which calls the compiled C code in Appendix F.2, establishes a connection to the server, and takes the raw ADC data from the C code; then it parses for frequency, RMS voltage, and phase data, and sends the data to the server.

```
#!/usr/bin/python

from __future__ import division

import unixSocketServer
import datetime
import sys
import subprocess
import os
import Adafruit_BBIO.GPIO as GPIO
# for socket connections, make sure lib is in same directory
import time
from lib import messageData
from lib import messageConfig
from lib import clientPMU
from lib import enum

ENUM = enum.ENUM




# for FFT and frequency/phase/rms analysis
from numpy.fft import rfft, irfft, fft
from numpy import argmax, sqrt, mean, diff, log, hanning, angle,
    pi
from time import time, sleep
import math
import cmath
from parabolic import parabolic, parabolic_polyfit

GPIO.setup("P8_12", GPIO.OUT)
GPIO.setup("P8_14", GPIO.OUT)
GPIO.setup("P8_16", GPIO.OUT)

def freq_from_fft(signal, fs):
```

```
35      """Estimate frequency from peak of FFT
36
37      """
38      # Compute Fourier transform of windowed signal
39      windowed = signal * hanning(len(signal))
40      f = rfft(windowed)
41
42      # Find the peak and interpolate to get a more accurate peak
43      i = argmax(abs(f)) # Just use this for less-accurate, naive
        version
44      true_i = parabolic(log(abs(f)), i)[0]
45
46      phase = angle(f[i])
47
48
49      # Convert to equivalent frequency
50      return (fs * true_i / len(windowed),phase)
51
52  # The fields that are set in the next two functions can be
        hardcoded in the
53  # __init__ function if they won't be changing.  This was a guess
         at what could be changing
54  def createConfigMessage():
55      configMsg = messageConfig.MessageConfig()
56      now = datetime.datetime.now()
57      configMsg.setSOC(now.year, now.month, now.day, now.hour, now
        .minute, now.second)
58      configMsg.setFRACSEC(
59              ENUM.LEAP_SECOND_DELETE,
60              ENUM.LEAP_SECOND_NOT_OCCURRED,
61              ENUM.LEAP_SECOND_PENDING,
62              ENUM.QUALITY_10n4,
63               463000, # This number gets divided by value of the
        field TIMEBASE
64      )
65      configMsg.setSTN("TEST ARS") # 16 Character max
66      configMsg.setFORMAT(
67              ENUM.FORMAT_INT,
68              ENUM.FORMAT_FLT,
69              ENUM.FORMAT_INT,
70              ENUM.FORMAT_RECT
71      )
72      configMsg.setPHNMR(3) # 1 Channel name per phasor
```

```
73      configMsg.setANNMR(1) # 1 Channel name per analog value
74      configMsg.setDGNMR(1) # 16 Channel names per digital value
75
76      chCount = 3 + 1 + 1*16
77      chNames = [
78              "VA",
79              "VB",
80              "VC",
81              "ANALOG1",
82              "BREAKER 1 STATUS",
83              "BREAKER 2 STATUS",
84              "BREAKER 3 STATUS",
85              "BREAKER 4 STATUS",
86              "BREAKER 5 STATUS",
87              "BREAKER 6 STATUS",
88              "BREAKER 7 STATUS",
89              "BREAKER 8 STATUS",
90              "BREAKER 9 STATUS",
91              "BREAKER A STATUS",
92              "BREAKER B STATUS",
93              "BREAKER C STATUS",
94              "BREAKER D STATUS",
95              "BREAKER E STATUS",
96              "BREAKER F STATUS",
97              "BREAKER G STATUS",
98      ]
99      for i in range(0, chCount):
100         configMsg.addCHNAM(i, chNames[i])
101
102     return configMsg
103
104 def createDataMessage():
105
106     dataMsg = messageData.MessageData()
107     dataMsg.setSOC(2006, 6, 6, 9, 0, 0)
108     dataMsg.setFRACSEC(
109             ENUM.LEAP_SECOND_ADD,
110             ENUM.LEAP_SECOND_NOT_OCCURRED,
111             ENUM.LEAP_SECOND_NOT_PENDING,
112             ENUM.QUALITY_NORMAL,
113             16817,
114     )
115
```

```
116     phsrInfo = [
117             ("VA", 3932, 7854),
118             ("VB", -7317, -12674),
119             ("VC", -7317, 12441),
120     ]
121
122     for i in range(0, len(phsrInfo)):
123         dataMsg.addPHASORS(phsrInfo[i][0], phsrInfo[i][1],
    phsrInfo[i][2], ENUM.FORMAT_INT)
124
125     dataMsg.setFREQ(350)
126
127     return dataMsg
128
129 os.chdir("/home/ring")  # TODO: figure out if this is still
    needed. Added because things weren't working on bootup, but
    probably wasn't the issue
130 # Address of test server was changing, could take in argument to
     make it easy to udpate
131 ip = "192.17.223.139"
132
133 sleep(5)
134 #Ethernet, check if ethernet is connected
135 p =  os.popen('cat /sys/class/net/eth0/carrier').readline()
136 sleep(0.1)
137 if int(p) == 1:
138   GPIO.output("P8_14", GPIO.HIGH)
139 elif int(p) == 0:
140   GPIO.output("P8_14", GPIO.LOW)
141 else:
142   print 'error'
143
144 configMsg = createConfigMessage()
145 configMsg.printNeatly()
146 dataMsg = createDataMessage()
147 dataMsg.printNeatly()
148
149 # Use UDP so you don't have to keep disconnecting and
    reconnecting to PMU
150 # Test.  TCP will work as well though
151 client = clientPMU.ClientPMU(ip, 4712, "UDP")
152 client.createSocket()
153 client.openConnection()
```

```
154
155 #i = raw_input("Start PMU Connection Tester and press <enter>")
156 client.sendMessage(configMsg.fullFrame())
157
158
159
160
161
162 # Create server object that will communication with client at
        the path input
163 server = unixSocketServer.UnixSocketServer("/tmp/testSocket")
164
165 # Start the server and only allow 1 request in the queue at a
        time.
166 # 1 Should be all we need since this will be on a board with
        only 1 PMU
167 # but it doesn't hurt to have the option to increase it just in
        case
168 server.startServer(1)
169 fs = 10000            # ADC sampling frequency on PRU
170 fPMU = 20           # number of frequency/phase/Vrms samples to
        return /second
171 sProc = int(fs/fPMU)     # size of data to process, in this case
        1000 samples at 20kS/s = 50mS
172 data = [0]*sProc       #data array for frequency/phase/rms calc
173 f = open('test.csv', "w") #debug
174 fnom = 60           #nominal frequency  - may need to read from
        file for 50Hz operation
175 tBase = 200000000      #PRU timebase = 200MHz
176
177 P = subprocess.Popen(["/home/ring/./ring_app"], stdout=
        subprocess.PIPE)
178 GPIO.output("P8_16", GPIO.HIGH)      # CS line on board
179
180 while True:
181   try:
182     # Will block program execution until an incoming connection
        is achieved
183     server.waitForConnection()
184     start = datetime.datetime.now()
185     #print 'hour = %s' %start.hour
186     i = 0
187     k = 0
```

```
188     accm = 0        #stores timebase accumulation measured against
        PPS error
189     lastPPScnt = 0    #temp for storing last PPS count
190     ppsShift = 0    #stores the calculated shift measured from
       PPS
191     lagcount = 0
192     sec = 0         # stores seconds of data
193     lastsec = 1
194     set = 0
195     ppsEnable = 0   # turn this off since GPS is not working
196     print "Receiving Data..."
197     ticks = int(server.readSample(10))
198     #print 'ticks count = %s' % ticks
199     pps = server.readSample(10)
200     #print 'pps count = %s' % pps
201     now = start     #TODO: this will come from GPS in the future
202     dataMsg.setSOC(now.year, now.month, now.day, now.hour, now.
       minute, now.second)
203     dataMsg.setFRACSEC(
204         ENUM.LEAP_SECOND_ADD,
205         ENUM.LEAP_SECOND_NOT_OCCURRED,
206         ENUM.LEAP_SECOND_NOT_PENDING,
207         ENUM.QUALITY_NORMAL,
208         0,
209     )
210     while True:
211       # Will read 4 bytes of data but can be changed should the
       need arise
212       data[i] = server.readSample(4)
213       if data[i]:
214#         s = str(data[i])
215#         f.write(s)
216#         f.write('\n')
217         # TODO: need to include real scaling for this - probably
        read from file
218         data[i] = float(data[i])/8191*3.3-1.65
219         i += 1
220
221         if i==sProc:
222           k += 1
223           i = 0
224           freq, phase = freq_from_fft(data, fs)
225           #print '%f Hz'   % freq
```

```
226             #print '%f phase'   % phase
227             rms = math.sqrt(sum([j*j for j in data])/len(data))
228             if rms*120 > 100 & set == 0:  #TODO: update for real
    scaling
229               GPIO.output("P8_12", GPIO.HIGH)
230               set = 1
231             if rms*120 < 100:        #TODO: update for real scaling
232               GPIO.output("P8_12", GPIO.LOW)
233               set = 0
234
235             #print 'Vrms = %s' % rms
236             df = int((freq - fnom)*1000)  #scale freq to mHz
237             #print 'delta f = %s' % df
238             if (df < 0):
239               df += 65536   # TODO: temp fix for setFreq not
    handling neg numbers
240             dataMsg.setFREQ(df)
241             #phasor sent in rectangular format
242             VA = cmath.rect(rms*120/3052*100000, phase)   #120 is
    just rough scaling, 3052/100,000 is from PHUNIT
243             dataMsg.setPHASORS(
244               (int(VA.real), int(VA.imag)),
245               (7283, 0),
246               (8283, 0),
247               ENUM.FORMAT_INT
248             )
249
250             #print 'shift = %s' % int(((k-1)*sProc/float(fs)+
    ppsShift)*1000000)
251
252             if (k-1)*sProc/float(fs)+ppsShift+sec > lastsec:
253               now = now + datetime.timedelta(seconds=1)
254               dataMsg.setSOC(now.year, now.month, now.day, now.
    hour, now.minute, now.second)
255               lastsec += 1
256
257             if (k-1)*sProc/float(fs)+ppsShift < 0:
258               fracsec = 1+ (k-1)*sProc/float(fs)+ppsShift
259             else:
260               fracsec = (k-1)*sProc/float(fs)+ppsShift
261 #           print 'ppsShift = %s' % ppsShift
262 #           print 'fracsec = %s' % fracsec
263             dataMsg.setFRACSEC(
```

```
264                 ENUM.LEAP_SECOND_ADD,
265                 ENUM.LEAP_SECOND_NOT_OCCURRED,
266                 ENUM.LEAP_SECOND_NOT_PENDING,
267                 ENUM.QUALITY_NORMAL,
268                 int(fracsec*1000000),      #1,000,000 comes from the
    timebase value set
269                 )
270
271             client.sendMessage(dataMsg.fullFrame())
272             #read ticks/pps every second
273             if k == fPMU:    # TODO: think about whether this
    should be fPMU
274 #DB removed for no PPS           ticks = server.readSample(10)
275 #               print 'ticks count = %s' % ticks
276 #               s = str(ticks)
277 #               f.write(s)
278 #               f.write('\n')
279 #DB removed for no PPS           pps = server.readSample(10)
280 #               print 'pps count = %s' % pps
281                 k = 0
282                 sec +=1
283                 print 'Seconds = %s' % sec
284                 print 'VRMS = %s' % (rms*120)
285                 print 'Frequancy = %s' % freq
286                 print 'Phase = %s' % phase
287                 if((pps > lastPPScnt & ppsEnable == 1)):
288                   ppsShift += (tBase - ticks/(pps - lastPPScnt))/
    float(tBase)
289                   lastPPScnt = pps
290
291 #               print 'ppsShift = %s' % ppsShift
292                 #TODO: need to think on this some more - should work
     for dropped PPS... I think
293                 #TODO: need to add code for lack of PPS pulse for
    certain amt of time - 30Sec I think from the spec
294                 #TODO: probably also need to have a flag for
    communication from the C Code
295
296 #             s = str(pps)
297 #             f.write(s)
298 #             f.write('\n')
299             #temp = data[i]
300
```

```
301         else:
302           lagcount +=1
303           if lagcount == 1000:
304             break
305 #     print "%2f last data" % temp
306       end = datetime.datetime.now()
307       diff = end - start
308       print "- - - - -"
309       print "%s samples" % i
310       print "%s seconds" % (diff.total_seconds())
311     except KeyboardInterrupt:
312       server.stopServer()
313       sys.exit(1)
314       GPIO.output("P8_16", GPIO.LOW)
315       GPIO.output("P8_14", GPIO.LOW)
316       GPIO.output("P8_12", GPIO.LOW)
```

code/testUnixSocketServer.py

REFERENCES

[1] "Model pl-403144 polymer lithium-ion battery product specification," AA Portable Power Corp, 860 S 19TH Street, A, Richmond, CA 94804, Datasheet.

[2] "150-ma low-noise low-dropout regulator with shutdown," Texas Instruments, Post Office Box 655303, Dallas, Texas 75265, Datasheet, June 2011.

[3] G. Coley and R. P. J. . Day, "Beaglebone black system reference manual," BeagleBone Black System, 12500 TI Blvd., Dallas, Tx 75243, Manuals, April 2013, rev A5.2.

[4] "Quad-channel, 5 kv isolators with integrated dc-to-dc converter," Analog Devices, One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A., Datasheet, 2012.

[5] J. Y. Cai, Z. Huang, J. Hauer, and K. Martin, "Current status and experience of wams- implementation in north america," *Transmission and Distribution Conference and Exhibition: Asia and Pacific*, 2005.

[6] A. Phadke and J. Thorp, "History and applications of phasor," in *Power Systems Conference and Exposition*, 2006.

[7] *Substation Communications: Enabler of Automation / An Assessment of Communications Technologies*, KEMA, Inc. Std.

[8] "New tools for keeping the lights on," July 2013, New York Times online.

[9] Y.-F. Huang, S. Werner, J. Huang, N. Kashyap, and V. Gupta, "State estimation in electric power grids: Meeting new challenges presented by the requirements of the future grid," *Signal Processing Magazine, IEEE*, vol. 29, no. 5, pp. 33–43, Sept 2012.

[10] R. F. Nuqui, "State estimation and voltage security monitoring using synchronized phasor measurement," PhD Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, July 2001.

[11] A. Phadke, "Synchronized phasor measurements — a historical overview," *Transmission and Distribution Conference and Exhibition*, 2002.

[12] Z. Zhong, C. Xu, B. Billian, L. Zhang, S. J. S. Tsai, R. W. Conners, V. A. Centeno, A. Phadke, and Y. Liu, "Power system frequency monitoring network (fnet) implementation," *Power Systems, IEEE Transactions*, vol. 20, no. 4, pp. 1914 – 1921, November 2005.

[13] R. Gardner, J. K. Wang, and Y. Liu, "Power system event location analysis using wide-area measurements," *Power Engineering Society General Meeting*, 2006.

[14] Y. Zhang and P. Markham, "Wide-area frequency monitoring network (fnet) architecture and applications," *IEEE Trans. on Smart Grid*, 2010.

[15] Z. Lin, T. Xia, Y. Ye, Y. Zhang, L. Chen, Y. Liu, K. Tomsovic, T. Bilke, and F. Wen, "Application of wide area measurement systems to islanding detection of bulk power systems," *IEEE Trans. on Power Systems*, 2006-2015.

[16] S. Qin, S. T. Cady, A. D. Dominguez-Garcia, and R. Pilawa-Podgurski, "A distributed approach to mppt for pv sub-module differential power processing," in *Proc. of the Energy Conversion Congress and Exposition*, 2013, pp. 2778–2785.

[17] "IEC appliance inlet c14 or c18, screw-on mounting, rear side, PCB terminals," Schurter Electronic Components, 447 Aviation Boulevard, Santa Rosa, CA, Datasheet, February 2014.

[18] L. Ada, "Adafruit ultimate GPS," Adafruit Industries, Manual, January 2015.

[19] "Simple, miniature single-cell, fully integrated Li-Ion / Li-Polymer charge management controllers," Microchip Inc., 2355 West Chandler Blvd. Chandler, AZ 85224-6199, Datasheet, 2007.

[20] K. Martin and B. Kasztenny, "IEEE standard for synchrophasor measurements for power systems," *IEEE Power & Energy Society*, p. 49, December 2011.

[21] "Adc141s626 14-bit, 50 ksps to 250 ksps, differential input, micro power a/d converter," Texas Instruments, Post Office Box 655303, Dallas, Texas 75265, Datasheet, January 2009.

[22] "Standard value decade for 1 percent resistors," Brannon Electronics, Inc., Datasheet.

[23] "Axial lead & cartridge fuses, 217 series, 5 x 20 mm, fast-acting fuse," Littelfuse, Tech. Rep., February 2015.

[24] "Sideview led, 5 mm diameter tinted diffused package," Vishay Inc., Datasheet 83043, Rev. 2.1, April 2013.

[25] "1590xxe enclosure," Hammond Manufacturing, Datasheet, enclosure molded from FRABS UL 94-V0 Plastic. Other materials / Colours Available Consult Factory. [Online]. Available: www.hammondmfg.com

[26] X. Zhang, "High precision dynamic power system frequency estimation algorithm based on phasor approach," Master Thesis, Virginia Polytechnic Institute and State University, January 2004.

[27] A. P. Meliopoulos, V. Madani, D. Novosel, and G. Cokkinides, "Synchrophasor measurement accuracy characterization," *North American SynchroPhasor Initiative Performance & Standards Task Team*, 2008.

[28] "Understanding FFT windows," LDS-Group, Application Note, 2003.